IBM DB2 Alphablox

Developer's Reference

Version 84

SC18-9435-02

IBM DB2 Alphablox

Developer's Reference

Version 84

SC18-9435-02

Note:
Before using this information and the product it supports, read the information in "Notices" on page 501.

Third Edition (March 2006)

This edition applies to version 8, release 4, of IBM DB2 Alphablox for Linux, UNIX and Windows (product number 5724-L14) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Using This Reference

This *Developer's Reference* is designed for use as a reference for developing applications. The focus is on using the DB2 Alphablox Tag Libraries.

There are six tag libraries in the DB2 Alphablox Tag Libraries:

- Blox Tag Library
- Blox Form Tag Library
- Blox Logic Tag Library
- Blox Portlet Tag Library
- Blox Report Tag Library
- Blox UI Tag Library

The Blox Tag Library is the core tag library, providing the tags for adding Blox related to data access and retrieval, data presentation, and application infrastructure in your JSP. This book provides referential information for each of the Blox in this Blox Tag Library. Following the sections on each of the Blox in the Blox Tag Library are sections for each of the other tag libraries. The only exception is the Blox Report Tag Library. This library is described separately in the *Relational Reporting Developer's Guide* since the Blox involved and the application development approach are different.

Java API associated with Blox and related objects are not described in this book, but can be found in the provided Javadoc documentation. The DB2 Alphablox Information Center comes with four sets of Javadoc information:

Javadoc Documentation	**Description**
Blox API	This is the Javadoc documentation for all Blox and related objects in the DB2 Alphablox Tag Libraries, except the Blox Report Tag Library.
Blox API Change List	This set of Javadoc documentation lists all new, changed, deprecated, or removed APIs by comparing the Blox API Javadoc documentation for this release and for the previous release.
Relational Reporting API	This is the Javadoc documentation for all Blox in the Blox Report Tag Library to support creation of reports from relational data sources.
FastForward API	This is the Javadoc documentation for the FastForward application framework.

Blox Properties, Methods, and Tag Attributes

Tag attributes for a Blox typically map to corresponding Blox properties. For example, DataBlox has a `query` tag attribute for specifying the query to retrieve data. This corresponds to the `query` property available on the DataBlox Java bean. Each property typically has a setter method and a getter method. For the `query` property, the associated methods are `getQuery()` and `setQuery(...)`. Some tag attributes take a boolean value to specify whether a certain state or feature is turned on or off. For example, for the `autoConnect` and `enableShowHide` tag attributes and properties, the corresponding methods are `isAutoConnect()` , `setAutoConnect(...)`,, `isEnableShowHide()` and `setEnableShowHide(...)`. Reference information for all methods are available in the Javadoc documentation.

Locating Tag Reference Information

The tags in the Blox Tag Library are organized based on the Blox they are associated with. For example, tags for DataBlox are in the Chapter 10, "DataBlox Tag Reference," on page 157 section. For the Blox Report Tag Library, see the *Relational Reporting Developer's Guide*. Each of the remaining tag libraries are described in its own section.

The Blox Tag Library chapters of the *Developer's Reference* are organized into the following sections whenever appropriate:

Section	Description
Overview	Provides a brief description of the Blox and a description of its user interface (if any).
JSP Custom Tag Syntax	Provides the syntax for the custom tag and tag attributes used to create a Blox.
Tag Attributes by Category	Lists all tag attributes on a Blox by category and associated custom JSP tag attributes.

The table of contents lists each of these sections, and the index lists individual APIs by their names.

Using the Tag Attribute Descriptions

The tags and tag attributes for each Blox are documented in their corresponding reference chapters. This section describes how each property description is organized.

Tag Attribute

A brief description of what the property does.

Data Sources

Lists the data source types to which this property applies (for example, Multidimensional).

Syntax

```
Lists the syntax of the custom tag library
```

where:

Argument	Default	Description

This table describes any arguments the tag attribute takes.

Usage

This section provides any usage notes relevant to the property and tag attribute.

Examples

This section shows example of using the tags or links to other examples.

See Also

This section lists cross references to related properties or topics.

Chapter 2. Overview of Blox and the Blox UI Model

This chapter describes the Blox categories, the Blox object model, and the Blox UI model. For conceptual information about DB2 Alphablox, see the *Developer's Guide*.

Blox Categories

DB2 Alphablox includes a set of Blox components for building powerful analytic applications. This section provides an overview of these Blox components by category as follows:

- User Interface Blox
- Data Blox
- Analytic Infrastructure Blox
- Blox UI Components
- Business Logic Blox
- FormBlox
- Relational Reporting Blox

User Interface Blox

These Blox provide visual components to support data navigation in grids and charts formats, supported by page filters, toolbars, a menu bar, and a data layout panel. Blox in this category includes:

- GridBlox: presents data in a table format.
- ChartBlox: presents data in charts.
- DataLayoutBlox: provides a data layout panel that allows users to move dimensions to desired row, column, or page filter axis.
- PageBlox: provides a page filter to the data presented in the grid and chart.
- ToolbarBlox: provides easy access to data navigation and analysis functionality with a click of the icons.
- PresentBlox: combines all the above user interface Blox in one container for better layout and communications among the Blox.
- ContainerBlox: the foundation Blox for all the user interface Blox. If you want to build custom user interface, you can start with a ContainerBlox.

The custom JSP tags for these Blox are available in the Blox Tag Library (`blox.tld`).

Data Blox

These Blox provide access to data sources. DataBlox, in particular, is needed for the user interface Blox to connect to and obtain the result set from the data source of interest. Blox components in this category includes:

- DataBlox: accesses the data sources and retrieves the result set for the user interface Blox.

- StoredProceduresBlox: allows you to create a connection to a relational database and prepare a stored procedure statement for use.
- ResultSetBlox: lets you to arbitrarily push a ResultSet into the associated DataBlox. You can also extend the normal functions associated with a JDBC data source by attaching a method to intercept queries in the DataBlox and to return an arbitrary ResultSet to the DataBlox.
- JDBCConnection bean: allows you to get information about an Alphablox relational data source, perform JDBC calls, or override properties of a JDBC data source defined in DB2 Alphablox.

The custom JSP tags for DataBlox and StoredProceduresBlox are available in the Blox Tag Library (`blox.tld`).

Analytic Infrastructure Blox

These Blox provide means to building the analytic infrastructure. Blox in this category includes:

- AdminBlox: provides programmatic access to information on the server, users, groups, roles, data sources, and applications set through the DB2 Alphablox Admin Pages.
- BookmarksBlox: allows you to programmatically create and manage bookmarks and dynamically set the bookmark properties.
- CommentsBlox: provides cell commenting (also known as cell annotations) as well as general page/application commenting functionality to your application.
- RepositoryBlox: provides a means for you to save and retrieve user and application properties stored in the DB2 Alphablox Repository.

The custom JSP tags for these Blox are also available in the Blox Tag Library (`blox.tld`).

Blox UI Components

The DHTML client is built on the Blox UI model with three distinct parts: the visual elements on a page (the Components), the Controllers that process events from the components, and Events that communicate state changes from the user interfaces and underlying application logic. These UI components are extensible, allowing you to extend beyond the out-of-the-box functionality the user interface Blox provide.

The tags in the Blox UI Tag Library (`bloxui.tld`) allow you to:

- customize components in the user interface such as adding items to the menu bar or toolbar or creating your own menu bar or toolbar
- customize layout such as adding empty columns/rows or moving row headers to specified position (the butterfly layout)
- add custom data analysis functionality that can be fully integrated into the user interface

The Blox UI Tag Library (`bloxui.tld`) is described in Chapter 23, "Blox UI Tag Reference," on page 367. For a list of all the components and their methods, see the com.alphablox.blox.uimodel.* packages in the Javadoc documentation.

Business Logic Blox

These Blox components let you add business logic to your application:

- MDBQueryBlox: enables OLAP queries to be built with one language regardless of the underlying server's query language
- MemberSecurityBlox: gives you the ability to hide members from unauthorized users
- TimeSchemaBlox: supports dynamic time series, such as showing data from the "last 3 months"

The custom JSP tags for business logic Blox are available in the Blox Logic Tag Library (`bloxlogic.tld`), described in Chapter 21, "Business Logic Blox and TimeSchema DTD Reference," on page 341.

FormBlox

These Blox are built on top of the Blox UI components to create form-based interface. A series of FormBlox are available to provide a familiar HTML form interface that allows users to select the data source, dimensions, members, or other options you provide to create personalized query.

The custom JSP tags for FormBlox are available in the Blox Form Tag Library (`bloxform.tld`), described in Chapter 20, "Blox Form Tag Reference," on page 305.

Relational Reporting Blox

A set of Blox designed to build interactive reports from relational data sources. For details, see the *Relational Reporting Developer's Guide*.

Blox Object Model

The Blox API consists of many Java™ objects, and you can programmatically access many of the objects through other objects. For example, PresentBlox has the methods `getDataBlox()`, `getPageBlox()`, `getGridBlox()`, `getChartBlox()`, `getToolbarBlox()`, and `getDataLayoutBlox()`, which are used to access the DataBlox, PageBlox, GridBlox, ChartBlox, ToolbarBlox, and DataLayoutBlox nested within. This section describes the overall object model in the Blox API. As with most object models, there are many paths you can use to navigate through the Blox API; this section describes the most basic and common access points to help familiarize you with the API.

Tip: For developers experienced in using API documentation generated by the Javadoc tool, it is a useful tool in learning about the object model. The Javadoc documentation for the DB2 Alphablox Blox API is available from the Help link on DB2 Alphablox Admin Pages. If you have DB2 Alphablox Javadoc documentation installed locally, it is located at:

- *<alphablox_dir>/system/documentation/javadoc/blox/index.html* if the online documentation is associated with an existing instance of DB2 Alphablox.
- *<doc_install_dir>/javadoc/blox/index.html* if your online documentation is a standalone installation.

This section covers the following topics:

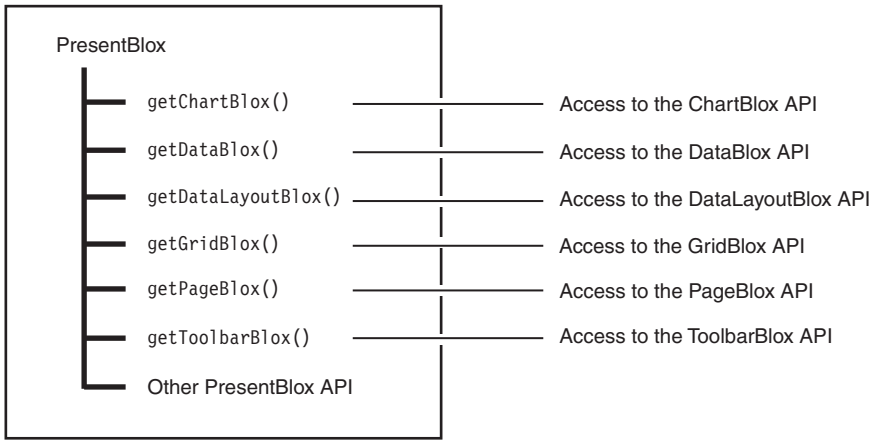
ContainerBlox— The Container for User Interface Blox

The ContainerBlox is base class for all user interface Blox. These Blox inherit the `bloxModel` property from ContainerBlox. Via the Blox's `getBloxModel()` method, you can access the UI model in effect for this Blox. Each Blox model consists of a header container and a body container, each contains a number of named standard components. You can use these names to find a component and customize it. This is discussed further in "Blox UI Model" on page 10.

PresentBlox—A Single Blox with Nested User Interface Blox

PresentBlox is a convenient way to display a chart, a grid, a data layout panel, and a toolbar all in a single Blox. Using a *nested Blox*, you can control all of the individual elements of each part that displays in a PresentBlox. Since you access each of the individual elements through the PresentBlox, you can think of the PresentBlox as a container for the other Blox, and those other Blox inside the PresentBlox container are known as nested Blox. Each Blox has properties that represent the state of the Blox, and you can access the nested Blox properties either by specifying values for the properties in the tag library used to create the Blox or by using the API to programmatically get and set the properties.

The following figure shows how you can access other Blox through PresentBlox:



Nested Blox

Some Blox can contain other, nested Blox. For example, ChartBlox and GridBlox (each of which can be a standalone Blox) are nested Blox within a nesting PresentBlox. DataBlox can be nested within PresentBlox, ChartBlox, or any of the Blox that take a data source. To apply a Blox-specific property to a nested Blox, add the nested tag. Nested Blox are accessed using the object model. Start with the outer Blox and then access the inner Blox calling the get method for the Blox you want (for example, `getDataBlox()`) to obtain access to the inner Blox object.

You can use the Blox Tag Library to create and access nested Blox. The following example, for an IBM DB2 OLAP Server or Hyperion Essbase data source, shows a DataBlox nested within a ChartBlox:

```
<blox:chart id="myChart"
...ChartProperty="ChartPropertyValue" >
   <blox:data
      dataSourceName="FinancialCube">
      query="!">
   </blox:data>
</blox:chart>
```

A PresentBlox typically contains multiple Blox that share one DataBlox:

```
<blox:present id = "profitPresent"
   height = "80%"
   width = "96%"
   dividerLocation = "0.60" >

   <blox:data
      dataSourceName = "QCC-Essbase"
      useAliases = "true"
      query = "!">
   </blox:data>

   <blox:chart
      chartType = "Vertical Bar, Side-by-Side"
      legend = "All Products"
      XAxis = "All Time Periods" >
   </blox:chart>

   <blox:grid
      bandingEnabled = "true">
   </blox:grid>

</blox:present>
```

If you have an explicit DataBlox to be used by multiple presentation Blox, you can use the DataBlox as a nested Blox via the `bloxRef` tag attribute:

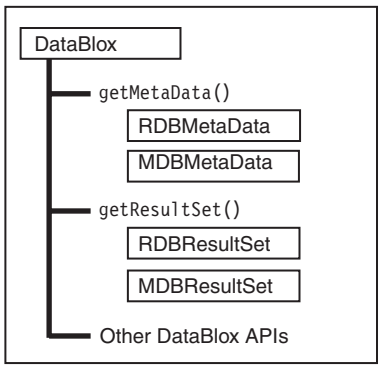
```
<blox:data id="FinancialCube"
   dataSourceName="FinancialCube">
   query="!" />

<blox:chart id="myChart"
...ChartProperty="ChartPropertyValue" >
   <blox:data bloxRef="FinancialCube" />
</blox:chart>

<blox:grid id="myGrid"
...GridProperty="GridPropertyValue" >
   <blox:data bloxRef="FinancialCube" />
</blox:chart>
```

For the syntax of each of the Blox custom tags, see the "JSP Custom Tag Syntax" section for a given Blox.

DataBlox—Access to Metadata and Result Sets

DataBlox provides access to data sources not only in terms of retrieving queries, but also in terms of searching through the metadata in the database (that is, the information about the data such as what members belong to a given dimension or what columns belong to a given table, what are the names of the dimensions or what are the names of the tables, etc.) and in searching through the data that is retrieved in a result set.

The following figure shows how you can access the metadata and result set objects through DataBlox. The metadata and result set objects each contain several objects.



To use the APIs associated with RDBMetaData and RDBResultSet, you need to import the com.alphablox.blox.data.rdb package in your JSP page:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

To use the APIs associated with MDBMetaData and MDBResultSet, you need to import the com.alphablox.blox.data.mdb package in your JSP page:

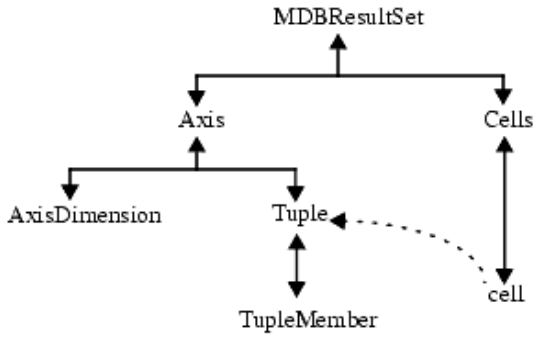
```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

Metadata and Result Sets

Result sets and metadata provide a way of walking the data in a hierarchical fashion. They are represented as objects with a rich set of APIs that give you full control of data presentation and interaction. You can access these objects through the `getMetaData()` and `getResultSet()` DataBlox methods.

The result sets objects involve actual data values from a query. Therefore, they have a restricted set of axes, tuples, dimensions, and members. Metadata objects do not need a result set from a query, and only involve the cubes, dimensions, and members (outline) of the data source. Generally speaking, you use MDB or RDB result sets if you are performing data source specific tasks such as calculations, writeback, and custom view. When what you do involves browsing outline or forming queries, you should use the metadata objects.

MDBResultSet: The following figure shows the object hierarchy of MDBResultSet. The direction of the arrows indicates whether you can reference the parent or child of an object. The dotted arrow means that once you get to the individual cell, you can find out its tuple, which allows you to access the axis it is on or a specific tuple member.



Note that a MDBResultSet object typically contains multiple axes and multiple cells, and each axis typically contains multiple tuples or dimensions. Therefore,

you have one method that returns an array containing all the axes, dimensions, tuples, or cells, and another method that returns one particular axis, dimension, tuple, or cell if you specify a 0-based index. For example, `getAxes()` returns an array containing all the axes in the result set, and `getAxis(0)` returns the first axis in the result set.

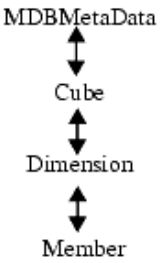
Some of the objects have types that give you easier access to the child object you want. For example, the Axis object has fields called `ROW_AXIS`, `COLUMN_AXIS`, `PAGE_AXIS`, and `SLICER_AXIS`. This allows you to easily access an axis of a specific type. Likewise, AxisDimension has types such as `ATTR_DIMENSION`, `MEASURES_DIMENSION`, and `TIME_DIMENSION` so you can easily access a dimension of a specific type.

RDBResultSet: The following figure shows the object hierarchy of RDBResultSet. The direction of the arrows indicates whether you can reference the parent or child of an object.

The RDBResultSet object contains a ResultColumn object that gives you information on the column type, column name, or position (0-based index) in the result set. The row iterator is an array of objects (`Object[]`) that lets you iterate through the rows to get the data.

Similar to the MDBResultSet object, an RDBResultSet object typically contains multiple columns. Therefore, you have one method that returns an array containing all the ResultColumn objects, and another method that returns one particular column. For example, `getColumns()` returns an array of result columns in this result set, and `getColumn(0)` returns the first result column within this result set.

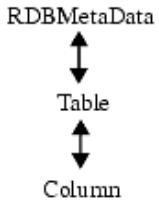
MDBMetaData: The following figure shows the object hierarchy of MDBMetaData. The direction of the arrows indicates whether you can reference the parent or child of an object.



Again, a MDBMetaData object may contain multiple cubes (in IBM DB2 OLAP Server or Hyperion Essbase, there is only one cube). Each cube typically contains multiple dimensions, and each dimension typically contains multiple members. As a result, you often have one method that returns an array containing all the cubes, dimensions, or members, and another method or methods that returns one

particular cube, dimension, or member if you specify a 0-based index. For example, `getCubes()` returns an array of cubes, and `getCube(0)` returns the first cube within the database this MDBMetaData object is describing.

RDBMetaData: The following figure shows the object hierarchy of RDBMetaData.



An RDBMetaData object may contain multiple tables, and each table typically contains multiple columns. As a result, you often have one method that returns an array containing all the tables and columns, and another method that returns one particular table or column if you specify a 0-based index. For example, `getTables()` returns an array of tables, and `getTable(0)` returns the first table within the database this RDBMetaData object is describing.

For a listing of the methods on each object discussed in this section, see "DataBlox Tag Attributes by Category" on page 160

Blox UI Model

The DHTML client is based on the Blox UI Model with three distinct concepts in the framework: components, controllers, and events. Components make up the visual elements on a page, such as buttons, edit fields, images, texts, toolbars, and menus. Controllers process events from the components, translating generic component behaviors such as `ClickEvent`, `RightClickEvent`, `DoubleClickEvent`, or `SelectedEvent` into application-defined action. Events communicates state changes from the user interface, the underlying application logic, and from the model itself to the components and controllers.

The following section provides a high-level overview of Components, Controllers and Events. These are common concepts and terms you will come across when you work with the Blox UI Tag Library or when you want to customize the behavior of a component. This general information will help you understand the UI model in the DHTML client and provide a foundation for customizing and extending the Blox UI model. Extending the Blox UI model is an advanced topic and is discussed in the *Developer's Guide*. Details on the objects and their associated methods are available in the DB2 Alphablox Blox API Javadoc documentation under the com.alphablox.blox.uimodel.* packages.

Component

Every visual user interface object in the Blox UI model descends from the Component base class. This model provides a number of core, basic user interface components such as Button, CheckBox, RadioButton, Edit (edit fields), ListBox, DropDownList, Menu, Menubar, Toolbar, ToolbarButton, DropDownToolbarButton, and ComponentContainer. All these components share a number of common properties and behaviors and are arranged in a hierarchy that provides both formatting control as well as central management of primitive components. The grouping of these components are made possible by the ComponentContainer, which allows grouping of components for the purposes of layout, behavior, and

style. For details on the components, their hierarchy, and their methods, see the com.alphablox.blox.uimodel.core package in the Javadoc documentation.

These UI components are further combined into compound components in the server's Blox Models. `BloxModel` is the base class for `GridBloxModel`, `ChartBloxModel`, `DataLayoutBloxModel`, `PageBloxModel`, and `PresentBloxModel`. It is used to represent the current visual state of ViewBlox-derived Blox (see the Blox object hierarchy in "ContainerBlox— The Container for User Interface Blox" on page 6).

Each Blox model consists of two named Containers:

- `ModelConstants.BLOXUI_HEADER` — Container which contains the toolbars and menu bar
- `ModelConstants.BLOXUI_BODY`— Container which contains the model(s) that provide the unique Blox functionality

Inside each of the main containers is a number of specifically named standard components that can be customized and/or removed. You can use these names to easily find a component during customization. All component names are available as constants in the interface class `ModelConstants`.

The following table lists the Blox UI model components. These are the components that made up the Blox user interface:

Blox UI Model Component	Description
Button	A push button component.
CheckBox	A checkbox component.
Component	The abstract base class for all UI model visual components. This class provides default behaviors and properties which are common across all visual components.
ComponentContainer	Generic container for UI model objects. The component container is used to group components for the purposes of layout, behavior, and style. For example, if you want three Buttons to line up horizontally from left to right, put them in a ComponentContainer and set the layout to horizontal. The order of component in the container is significant and affects the display order. A component can only exist exactly once in exactly one container. If a component is added to a different container, it is automatically removed from the previous container.
Controlbar	The base class for controlbars (menus and toolbars) that can be contained in a ControlbarContainer.
ControlbarContainer	The container for Controllbar.
DateChooser	This component extends the Edit component by adding a calendar icon next to the edit field. Clicking the icon launches a calendar widget for selecting a date to populate the edit field.

Dialog	Dialogs are floating containers that are used to collect input from and/or show status to users. Create a dialog and then populate the dialog with components such as Buttons, CheckBoxes and RadioButtons (among other) to present users with option lists or to make a decision.
DropDownList	A DropDownList consists of a single displayed option with a list of other choices. Only one choice may be selected at a time. Use a DropDownList when space is limited and the constant display of possible choices is not required.
DropDownToolbarButton	The DropDownToolbarButton has both a drop-down list of selections as well as an action button to invoke the currently displayed drop down list. The control generates a ClickEvent when the selection is changed in addition to when the action button is clicked.
Edit	Edit field component. An Edit component allows the user to enter and modify one or more lines of text. Text can be copied, moved, and inserted into the edit field using the standard user UI mechanisms.
GroupBox	GroupBox component provides a titled container for dialogs and other models. The GroupBox is primarily used to group components in dialog boxes. For example, if there are a number of components dedicated to setting options for a chart, then these can be grouped together in a GroupBox with the title “Chart Options.” When RadioButton components are contained inside of a `GroupBox`, all unnamed RadioButtons in a named group will become automatically grouped. Pressing one radio button will unselect others in the group.
Image	An Image component used to display any GIF, JPEG, or other compatible image. The image will generate a ClickEvent when clicked.
ListBox	A ListBox component that creates a selection list.
Menu	A `Menu` component consisting of MenuItems and other Menus. Menus inside of Menus will be treated as submenus with the appropriate submenu behavior. MenuItems will display as selections and will generate a ClickEvent when selected. By default, a MenuItem’s name is used to construct a handler method in controllers. For example, a MenuItem with the name ″drillDown″ will map to a method called ″actionDrillDown″ in controllers. All new menus are assigned a vertical layout by default.
Menubar	A Menubar component.

MenuItem	A MenuItem component. This is a selectable item in menu.
MessageBox	MessageBox dialog. The MessageBox dialog provides a convenient way to present simple information to the user. It also provides a simple mechanism to collect YES/NO and OK/Cancel responses from users.
RadioButton	RadioButton component. Typically RadioButtons are grouped together so that only a single RadioButton in a named group can be selected at any one time. Selecting a RadioButton in a group of RadioButtons will automatically unselect the currently selected button in that group. GroupBoxes can be used to automatically group together sets of RadioButtons.
Spacer	Spacer component, used to add fixed height and/or width spacing between components.
SpinnerButton	Spinner component, used to accept integer input from the user and provide buttons to increase/decrease the value. You can set the initial value, increment, and low and high values.
SplitterContainer	SplitterContainer component, used to display two components with a splitter bar between them that the user can adjust. Use either the HorizontalLayout or the VerticalLayout to control the orientation of the splitter.
Static	Static component, used to display simple static text such as instructions, labels, or values, where interaction is not needed. You may attach a ClientLink to a static component in order to make it respond to a user selection. The component's title field is used to store the Static text.
StaticImage	Component to render a static image which does not respond to user input. For images that will generate a ClickEvent, use the Image component.
TabbedContainer	Container window with tabs for all child containers. This container is used to display a list of tabs corresponding to all child containers. A typical use would be to implement a tabbed dialog box. This container can only contain other component containers. The style attached to this container is applied to the tabs. The title of the child container is used for that container's tab label. The selection state of the child containers is used to determine the selected tab. If no child containers are selected, then the first container is automatically selected. If multiple child containers are marked as selected, then the first one encountered is considered selected.

	The order in which the child containers are added to the tabbed container determines the tab order. For top and bottom (horizontal), the first container is on the right. For left and right (vertical), the first container is on the top.
Toolbar	Toolbar component used in conjunction with ControlbarContainer to display toolbars.
ToolbarButton	ToolbarButton component. Toolbar buttons can be used anywhere in the component model, but they are primary designed to work in ControlbarContainers. The name of the component is used to construct the image name with a `.gif` extension.

Events

The DHTML client creates events as JavaScript™ objects that can be created, sent, and intercepted by client-side JavaScript code. The DHTML client does not have any domain knowledge such as whether an event is data drill up or drill down, nor does it understand dimensions or members. All of the domain-specific logic and information is stored on the server. The DHTML client only knows when an object such as a menu item or a help button is clicked. It is only aware of a simple set of events such as single click, double click, right click, scroll, drag and drop, selected, unselected, selection changed, contents changed, and closed.

Controller

When users press a `Button` or select a `MenuItem` from a `Menu`, the controller is responsible for calling and executing the action associated with this user event. The `Controller` class is the base class for all model component controllers. Controllers can be attached to any model object that are derived from Component.

Server-Side API and Client-Side API

The APIs for developing applications using the DHTML client in DB2 Alphablox are available on the server-side, where a developer accesses them through Java calls (for example, in a Java scriptlet on a JSP page). The reason the Java APIs are called *server-side* APIs is because the code executes on the server before it is sent to the browser.

Executing code on the server is often more efficient, and also makes it easier to create web pages that work correctly on multiple browsers. The DHTML client is designed to keep the client and the server in sync without page refreshing. When you execute code on the server, only affected areas in the Blox UI is refreshed, not the whole page.

There are also times when you want to use the DHTML client's Client API for tasks that are best handled on the client. These are called client-side APIs because they are interpreted by the browsers. Often times you want to call some server-side code to change Blox properties on the server via some JavaScript code on the client when a user clicks a button or link on the page.

The DHTML client has a relatively straightforward API on the client-side. See Chapter 26, "Client-Side API," on page 437 for more information.

Chapter 3. General Blox Reference Information

This section provides general reference information that applies to all Blox. For information on tag attributes common to all Blox, see Chapter 4, "Common Blox Tag Reference," on page 29.

Tips for Working with Blox

Before working with Blox, note the following key points:

- The API change list Javadoc documentation includes a listing of deprecated Blox methods and properties, the release in which they were deprecated, and their replacements. When working with existing applications, refer to this Javadoc documentation to determine necessary changes to method and property names.
- If an invalid property is used on a Blox, the JSP file will not compile successfully.
- All Blox properties (and their corresponding tag attributes) are case-sensitive. With a few exceptions, the property names follow the Java bean naming convention: begin the name with a lowercase letter, each new word or phrase in the name begins with an uppercase letter (for example, `dataSourceName`).
- All URLs that run through DB2 Alphablox are case-sensitive.
- To override a default or inherited value for a specific Blox, include the property keyword and local value in the custom tag used to create the Blox. For example, the following attribute within the `<blox:chart>` custom tag causes ChartBlox to display a pie chart:

  ```
  chartType="Pie"
  ```

Working with Different Data Sources

DB2 Alphablox includes a Data Manager and associated data adapters that provide support for:

- browsing a collection of pre-configured, named connections to application data sources
- exposing the available databases within each data source
- publishing the compatible query types for a specific data source
- allowing the traversal of the data source metadata
- managing data source connections for user sessions
- translating query objects into the underlying native query language
- executing queries against a data source
- interrogating a query result set's data and schema

- modifying a result set by pivoting, expanding, and drilling

DB2 Alphablox data adapters can access and retrieve data from relational databases, multidimensional databases, and DB2 Alphablox cubes. (DB2 Alphablox cubes transform data from relational databases into multidimensional format).

Most Blox properties and methods apply to all types of data sources. For those that do not, there is a section in the API description stating which data sources a particular API works with (for example, all, multidimensional, relational, IBM® DB2 OLAP Server™ and Hyperion Essbase only, etc.).

Blox in a JSP File

This section describes the components of a JSP page that contains Blox. It shows a sample JSP file and then describes the different sections of it.

Sample JSP File Containing Blox

The following code listing shows a JSP file containing all the elements necessary to render a Blox on a page.

```
<%-- Import any packages used in scriptlets --%>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.data.*" %>
<%@ page import="com.alphablox.blox.data.mdb.*" %>

<%-- Import the Blox custom tag libraries --%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<%-- Set the UTF-8 Charset--%>
<%@ page contentType="text/html; charset=UTF-8" %>

<%-- Create the Blox --%>
<blox:present
   id="regionsBlox"
   visible="false"
   width="650"
   height="350"
   splitPane="false"
   visible="false">

   <blox:data
      dataSourceName="TBC"
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year, Scenario)
         Qtr1 Qtr2 <CHILD Scenario Sales !"
      useAliases="true"
      selectableSlicerDimensions="Market" >
   </blox:data>

   <blox:grid
      bandingEnabled="true" >
   </blox:grid>

   <blox:chart
    chartType="Vertical Bar, Stacked" >
   </blox:chart>

</blox:present>
<!-- HTML and JavaScript Elements -->
<html>
<head>
<title>Sample Blox JSP File</title>

<%-- Insert the Blox header --%>
```

```
<blox:header />

<%--Insert some JavaScript, if needed (with or without any
Blox APIs)--%>
<script language="JavaScript">
</script>

</head>
<body>

<%-- You can include scriptlets or JavaScript containing
     Blox APIs as needed --%>
<p>Put the Blox here <br />

<%-- Display the Blox --%>
<blox:display bloxRef="regionsBlox" />
</p>

</body>
</html>
```

Package and Tag Libraries Imports

This section is usually at the top of the JSP file. The package import statements are only necessary if you are using any APIs in those packages. The tag libraries import section is required to use any of the Blox tag libraries.

```
<%-- Import the Blox Tag Library --%>
<%@ taglib uri="bloxtld" prefix="blox"%>

<%-- Import the Blox UI Tag Library --%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<%-- Import the Blox Form Tag Library --%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>

<%-- Import the Blox Logic Tag Library --%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<%-- Import the Blox Reporting Tag Library --%>
<%@ taglib uri="bloxreporttld" prefix="bloxreport"%>

<%-- Import the Blox Portlet Tag Library --%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet"%>
```

Also use this section to import any Java packages used in Java API calls. The Java APIs might be called from a scriptlet on the JSP page. For example:

- If you are using the MDBMetaData object or the MDBResultSet object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.data.mdb.*" %>
  ```

- If you are using the RDBMetData object or the RDBResultSet object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.data.rdb.*" %>
  ```

- If you are using the BookmarksBlox and its associate objects, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.repository.*" %>
  ```

- If you are using the Comment object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.comments.*" %>
  ```

- If you are using the server-side Event Filter object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.filter.*" %>
  ```

- If you are using the StoredProcedure object, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.data.rdb.storedprocedure" %>
  ```

- If you are using the <jsp:useBean> tag to create Blox rather than using the Blox tag libraries, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.*" %>
  ```

- If you are using the Blox UI Model API, you will need the following import statement:

  ```
  <%@ page import="com.alphablox.blox.uimodel.*" %>
  ```

If an import statement is needed in order to use an API, this information is listed in the beginning of each of the API section.

Adding Content Type Character Set Declaration

To ensure proper character set encoding, add the following line to your JSP file to set the character set:

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

This is particularly important if you are running on a foreign language system that uses double-byte characters, such as Japanese and Chinese.

Blox Creation Tags

You can place your custom tags for creating the Blox anywhere on your JSP page, but it is a good idea, and can make your code cleaner looking and more readable, to place them before the HTML sections of the page. This helps separate the application logic from the display elements of your page. If you do place the Blox tags before any HTML elements, you also have to set the `visible` property to `false`, and then use the <blox:display> tag to actually display the Blox on your page. For details on the <blox:display> tag, see "The <blox:display> Tag" on page 22.

<blox:header> Tag in the HTML <head>

The <blox:header> tag should go in the HTML <head> section of your page. It is required for pages rendered in the DHTML mode to display properly as DB2 Alphablox substitutes the proper theme and style information in the header for the Header tag. It also adds a few lines of code that manage file caching for pages. More importantly, it provides the foundation for communications services between the client and the server, making it possible to execute server-side code from JavaScript objects on the client.

For more information on additional tag attributes for specifying page URL and context path for portlet integration or client bean registration, see the "The <blox:header> Tag" on page 23.

Tip: The <blox:header> tag must come before the Blox is set to display in your JSP file; that is, it either must come before a Blox creation tag with a `visible` property of `true` (the default) or it must come before the <blox:display> tag makes the Blox visible on the page.

Scriptlets Containing Blox APIs

You can put any Java code you want anywhere within a JSP page, and the code is executed on the server before the page is sent to the user. The Java code can use Blox APIs in a scriptlet or it can use anything available in Java or in the

environment in which your web application server is running. If the Java code uses Blox APIs, the Blox definition to which the code scripts must come before the scripting code. To place Java code on your JSP page, place any valid Java code between the following sets of characters:

```
<%
%>
```

The JSP engine recognizes this as Java code and compiles and executes it (it will only be compiled the first time the page is loaded or if the page has changed since the last compile). For example, the following scriptlet uses the server-side ChartBlox APIs and standard `out.write()` Java method to print the values of the `rowSelections` and `columnSelections` properties to the Java console:

```
<%
String RowSelections = mypresent.getChartBlox().getRowSelections();
String ColumnSelections =
       mypresent.getChartBlox().getColumnSelections();
       out.write("The value of columnSelections is:" +
                           ColumnSelections);
       out.write("The value of rowSelections is:" +
                           RowSelections);
%>
```

Note: If you are running an application server and the DB2 Alphablox console is not available, you can use other techniques such as writing the output to a log file, or using the UI Model's MessageBox to display the output during development.

Note: JSP technology includes many techniques for scripting. For details on different ways to script in a JSP file, see a JSP reference book.

How Scriptlets are Evaluated—Inside the Tag versus Outside the Tag

The Blox tags are only evaluated the first time a page is loaded for a user session, while everything outside of the tag is evaluated each time the page is loaded. When the Blox tag is evaluated, the state of all the properties at that time is rendered to the page. If you have scriptlets inside a Blox tag, the code in the scriptlet is executed before the Blox is rendered, therefore any changes that the code might make to a property will be reflected in the Blox rendered to the page.

Because code outside the Blox tags is evaluated after the Blox is evaluated and rendered to the page, any changes to properties in a scriptlet outside the Blox tag will not show up in the Blox on the page until the page is reloaded. Therefore, if you put a scriptlet which changes the value of a Blox property inside the Blox tag, it is evaluated before the Blox is rendered to the page, so the changes appear on the initially rendered page; if you put a scriptlet which changes the value of a Blox property outside the tag, it is evaluated after the Blox is rendered to the page, so the property change is not reflected in the Blox until the page is reloaded.

Sometimes you might need to perform some logic in a scriptlet to determine how to set a property, but you also want that logic to execute each time the page is loaded (not just the first time the page is loaded). Putting the code inside the Blox tag would execute the code for the first load of the page in a user's session, but subsequent page refreshes in that session will not execute the code in the Blox tag. In this situation, you can set the `visible` property in the Blox tag to `false` and put the code to set the property in a scriptlet outside the Blox tag. Then, later on in the page, use the `<blox:display>` tag to display the Blox on the page. This technique

results in properties you set outside the Blox tag to be reflected in the Blox that is displayed on the users page. The following pseudo code demonstrates this technique:

```
<!--The Blox tag creates the Blox, but since the visible
    property is set to false, the Blox is not yet sent to
    the browser -->
<blox:grid
    id="myBlox"
    visible="false"
    ....
    ...the rest of the tag definition >
</blox:grid>

<%
   // this scriptlet executes some code to set a property
   // (for example, based on who the user is)
   // Because it is outside the tag, it will execute when each
   // time the page is loaded
%>
<!--Use the display tag after the code has executed.
<blox: display
      bloxRef="myBlox"
      visible="true" />
```

JavaScript Code Containing Blox APIs

Anywhere within the HTML section of your JSP page, you can place JavaScript elements using the HTML `<script>` tag.

Tip: The best practice for putting `<script>` tags in HTML pages is to locate them between the `<head>` tags, the `<body>` tags, or between the `<html>` tags if there are no `<head>` or `<body>` tags. The exception is if the `<script>` tag is writing out the `<head>` or `<body>` tags.

HTML and JavaScript Elements

Of course, you can put any HTML or JavaScript elements on your JSP page and it will just be passed on through to the browser. The JSP engine ignores all HTML and JavaScript elements.

URL Attributes

DB2 Alphablox provides several URL attributes as a convenient way to change the render mode, saved state, and theme for an application. URL attributes can be added to the application's URL to define runtime processing. URL attributes take the following form:

```
attribute=value
```

For example, the `render` attribute specifies the format into which DB2 Alphablox renders a page before it is delivered to a client browser. The following attribute specifies that the page is to be delivered in DHTML:

```
render=dhtml
```

To add a single attribute to a URL, append the attribute at the end of the URL preceded by the "?" symbol, as in the following example:

```
http://serverName/App_Context/MyApp.jsp?render=dhtml
```

To add other URL attributes, append them with the & character, as follows:

```
http://serverName/App_Context/MyApp.jsp?theme=financial&render=dhtml
```

This section describes the following valid URL attributes:

- "render" on page 21
- "theme" on page 21

Tip: URL attributes are case-sensitive; they are all lowercase.

For an example of using a URL attribute and RepositoryBlox to load a saved application state, see the Repository `restoreApplicationState()` method in the Javadoc documentation.

render

`render=`*`string`*

Specifies the delivery format for all Blox on this application page. Note that the render property on a Blox takes precedence over this attribute.

Possible values include the following:

`dhtml`	Render in fully interactive DHTML format (the default). Requires the `<blox:header />` tag in your JSP page.
`printer`	Render in a format suitable for printing. Requires the `<blox:header />` tag in your JSP page.
`xls`	Render a format suitable for export to Microsoft® Excel, and set the MIME type to XLS. In order for the MIME type to be set so the page opens in Excel, you must put the `<blox:header />` tag in your JSP page. For information on the `<blox:header />` tag, see "<blox:header> Tag in the HTML <head>" on page 18.
`xml`	Render in XML format.

For more information on delivery formats, see "Render Formats Available to the DHTML Client" of the *Developer's Guide*.

Important: Using the render attribute without the `theme` attribute causes DB2 Alphablox to use the default theme and automatic browser detection.

theme

`theme=`*`themeName`*

Used in `dhtml` render mode. Specifies the theme to use when rendering this page. If the theme name is default, or if the attribute is not used, DB2 Alphablox automatically selects the most appropriate theme based on browser type, browser version, client operating system, and rendering format.

Note: For the `theme` attribute to be recognized, the following line must be added between the `<head>` and `</head>` tags in the HTML part of the JSP page:

```
<blox:header />
```

Valid values for the `theme` attribute are `coleman` (default) and `financial`. The coleman theme has a grey and blue tone, whereas the financial theme has a pale green tone.

Data Type Mappings

The following table shows how Alphablox data types map to JDBC and Java data types. The data ranges of Java types may differ from the ranges supported by a specific database.

JDBC Type	Alphablox Type	Java Type	Java Range
TINYINT	IntegerOperand	int	32-bit signed two's-complement integers
SMALLINT	IntegerOperand	int	32-bit signed two's-complement integers
INTEGER	IntegerOperand	int	32-bit signed two's-complement integers
BIT	boolean	boolean	true or false
BIGINT	LongOperand	long	64-bit signed two's-complement integers
REAL	FloatOperand	Float	32-bit IEEE 754
FLOAT	DoubleOperand	double	64-bit IEEE 754
DOUBLE	DoubleOperand	double	64-bit IEEE 754
NUMERIC	CurrencyOperand	double	64-bit IEEE 754
DECIMAL	CurrencyOperand	double	64-bit IEEE 754
CHAR	String	String	
VARCHAR	String	String	
LONGVARCHAR	String	String	

Note: Note the following about data type mappings:

- The content of a specific Java data type may also differ from that of a database. For example, Java date values begin with the year 1900; most databases accommodate earlier date values.
- JDBC data types may not map directly to your RDBMS. For help in this area, contact the vendor of your JDBC drivers.

The <blox:display> Tag

The `<blox:display>` tag references a Blox that has already been created and displays it wherever you place the tag. The most common use of this tag is to, after some processing logic is done, display a presentation Blox that already exists but has its `visible` attribute set to `false`. Place the `<blox:display>` tag where you want your Blox to display on your HTML portion of your JSP page.

```
<blox:display
    bloxRef="myPresentBlox" />
```

or

```
<blox:display
    blox="<%=myBlox %>" />
```

- Since this is an empty tag with no elements, it should be closed using the shorthand notation as shown above. Depending on the application server, you may get an error if you close it using a `</blox:display>` closing tag.
- Before using the `<blox:display>` tag, the Blox to which it refers (the value of the `bloxRef` or `blox` attribute) must already be instantiated. The instantiation occurs either if the Blox was previously created in another page or if the Blox comes before the `<blox:display>` tag in the JSP file.

- The `bloxRef` attribute lets you reference the name of a Blox. The `blox` attribute lets you reference an actual Blox object. For example, if you have the following GridBlox:

  ```
  <blox:grid id="mygrid" visible="false" />
  ```

  You can later display it using either attribute:

  ```
  <blox:display bloxRef="mygrid" />
  ```

  or

  ```
  <blox:display blox="<%= mygrid %>" />
  ```

  If you have the following GridBlox:

  ```
  <blox:grid id="mygrid" bloxName="grid1" visible="false" />
  ```

  You can later display it using either of the following code:

  ```
  <blox:display bloxRef="grid1" />
  ```

  or

  ```
  <blox:display blox="<%= mygrid %>" />
  ```
- If both `bloxRef` and `blox` attributes are specified, the value for `bloxRef` is ignored.

Tag attributes available to the `<blox:display>` tag are as follows:

```
<blox:display
    blox="<%=myBlox=>"
    bloxRef="myPresentBlox"
    render="dhtml"
    width="600"
    height="800" />
```

Any attributes that you set on the `<blox:display>` tag override any of those attributes that might be set in the tag it references. The above example shows a `<blox:display>` tag that will display a previously defined Blox whose `id` is `myBlox`, overriding the original `render`, `width`, and `height` settings with the ones specified in the display tags.

Note: The `<blox:display>` tag cannot reference a relational reporting Blox. Relational reporting Blox are Blox that support ReportBlox for producing interactive reports from relational data sources. The usage and reference material for those Blox are documented in the *Relational Reporting Developer's Guide*.

The <blox:header> Tag

The `<blox:header>` tag is required to be placed in the HTML `<head>` section of your page for Blox to be rendered properly in the `dhtml` mode. It:

- Tells DB2 Alphablox to include the proper theme and style information in the header for the Header tag
- Creates the appropriate BloxContext, BloxRequest, and BloxResponse objects for the given container request and response (for example, HTTP request and response, or portlet request and response).
- Provides the foundation for communications services between the client and the server, making it possible to execute server-side code from JavaScript objects on the client.

BloxContext contains references to all Blox in a session. It is created after the <blox:header/> tag is encountered in an JSP page, and there is only one BloxContext per session.

In cases such as portals or other proxy front-ends where the servlet request URI does not reference back to DB2 Alphablox or the application's context path, the <blox:header> tag has two attributes—pageURL and contextPath—that allow you to explicitly specify the page URL and application context path.

The <blox:header> tag also allows you to register your custom bean using its <blox:clientBean> inner tag. This registers the server-side bean with the DB2 Alphablox programming framework and makes the bean's methods available on the client. You can then invoke the bean's server-side method from client-side JavaScript. The following example shows the registration of a custom bean called myBean and its method called changeColor.

```
<blox:header>
   <blox:clientBean name="myBean" bean="<%= myBean %>">
      <blox:method name="changeColor" />
   </blox:clientBean>
</blox:header>
```

The bean name will be added to the BloxContext as an attribute. This is similar to calling the setAttribute() method on the BloxContext object:

```
bloxContext.setAttribute("myBean", myBean);
```

You then can call the changeColor method on the server using the client API's callBean() method:

```
myBean.changeColor("red");
```

If you do not have the bean, and only have a reference to it, you can still register it without specifying the bean attribute and invoke the bean's server-side method from client-side JavaScript. The following example shows a bean registration without the bean attribute.

```
<blox:header>
   <blox:clientBean name="myBean">
      <blox:method name="changeColor" />
   </blox:clientBean>
</blox:header>
```

The <blox:header> tag also provides a theme tag attribute. This attribute lets you specify the theme to use for this page. DB2 Alphablox determines which theme to use for a page based on the following priority:

1. URL parameter. The theme specified by appending the theme=*themeName* URL attribute when the page is called overrides other theme settings.
2. The <blox:header> tag. The theme setting at this page-level overrides the setting at the application-level or system-level.
3. The application's theme setting, specified through the DB2 Alphablox Admin Pages in the application's definition.
4. Default theme for this DB2 Alphablox instance, specified through the DB2 Alphablox Admin Pages, under the General Properties section's System link.

Tip: The <blox:header> tag must come before the Blox is set to display in your JSP file; that is, it either must come before a Blox creation tag with its visible attribute set to true (the default), or it must come before the <blox:display> tag that makes a Blox visible on the page.

Note: See "The <blox:bloxContext> Tag" on page 25 if you only want to create the appropriate BloxContext, BloxRequest, and BloxResponse objects without the themes or client-side JavaScript code.

The <blox:bloxContext> Tag

The `<blox:bloxContext>` tag is similar to the `<blox:header>` tag in that it creates the appropriate BloxContext, BloxRequest, and BloxResponse objects based on the type of request and response given. However, it does not put any rendering JavaScript or unnecessary themes on the page. An example of the use of this tag is when you do not have a Blox on a certain JSP page but need to access a Blox created by another JSP page to dynamically change its attributes. Since both the `<blox:bloxContext>` tag and the `<blox:header>` tag attempt to declare the same variables, they cannot coexist in a JSP page.

The <blox:session> Tag

This tag lets you synchronize the creation of the DB2 Alphablox session. This is useful when you are using framesets or iframes with Tomcat, where you do not want multiple session cookies sent to the browser, or you do not want to use the `<blox:header/>` tag on each JSP in the frames to put unnecessary JavaScript and styles on pages that do not have a Blox. This tag has an optional `key` attribute that should be unique for the application or browser session (that is, the frameset session ID). If the `key` attribute is not specified, a DB2 Alphablox session will be created and the session ID cookie will be returned. If a unique identifier is passed, it will prompt DB2 Alphablox to check if a DB2 Alphablox session has already been created with the key. For example:

```
<blox:session key="<%=session.getID() %>" />
```

or

```
<blox:session key="<%=request.getParameter( "syncKey" ) %>" />
```

where the `syncKey` is passed in when this page is called.

If a non-unique or invalid key is specified, you will have unexpected or undesired results such as data showing up in wrong sessions or session expired messages.

Note: For IBM WebSphere or BEA WebLogic, the key defaults to the J2EE session ID. Therefore there is no need to use this tag.

The <blox:logo> Tag

This tag adds a ″DB2 Alphablox″ logo with a hyperlink link to DB2 Alphablox product Web site. All that is required is the following:

```
<blox:logo />
```

Resource Bundle Related Tags

Resource bundles let you develop applications that can be localized and translated into different languages. Resource bundles contain key-value pairs, and the keys are always of String type. The keys are used to identify the localized and translated resource appropriate for the current user's locale. The detail for resource bundles is available in the Javadoc documentation for ResourceBundle at http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html.

The Blox Tag Library provides three wrapper tags on the standard Java ResourceBundle class for you to access a locale-specific resource from the resource bundle. The locale is based on Blox context (`BloxContext.getLocales()`), which means the locale does not change during the same session even if subsequent requests have different locale settings.

The basic usage of the tags is as follows:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<blox:resourceBundle bundle="mypackage.mystrings" />
<blox:message key="messageKey1"/>
<blox:message key="messageKey2">
   <blox:messageArg value="<%= argValue %>"/>
</blox:message>
```

where `mystrings` is the base name of the bundle to load, and the `WEB-INF/classes/mypackage/mystrings.properties` file contains the following entries:

```
messageKey1 = text for message key 1
messageKey2 = text for message key 2 containing the argument value {0}
```

The sample FastForward application is a working example of how these tags are utilized to support multiple locales in a custom application.

The <blox:resourceBundle> tag

This tag loads the specified resource bundle and sets that bundle as the default bundle for the `<blox:message>` tag. The bundles are typically loaded using the JSP page's application class loader, which allows resources to be located under an application's `WEB-INF/classes` and `WEB-INF/lib` directories. Bundles of the same group share a base name, followed by an underscore ("_") and then the locale. See the Javadoc documentation for ResourceBundle.getBundle() at http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html for details.

The `resourceBundle` tag has the following attributes:

Argument	Required	Description
`id`	No	The name of a JSP script variable to declare. The type of this variable is java.util.ResourceBundle.
`bundle`	Yes	The base name of the resource bundle to load.
`locale`	No	The locale to use to locate the resource bundle. If not specified, the locale is determined from `BloxContext.getLocales()`.
`path`	No	The directory root to use to locate the resource bundle files. If not specified, the JSP page's application class loader will be used.

The `<blox:resourceBundle>` tag is an empty tag and therefore should be added using the shorthand notation:

```
<blox:resourceBundle bundle="mypackage.strings"/>
```

Depending on the application server, you could get an error if you close it using a `</blox:resourceBundle>` closing tag.

The <blox:message> Tag

The `message` tag is used to display the resource identified by the specified key. If the `id` attribute is not specified, then the default resource bundle for the JSP page is used.

The `message` tag has the following tag attributes:

Argument	Required	Description
`id`	No	The id of the resource bundle to locate the key. If not specified, the default resource bundle is used.
`key`	Yes	The name of the resource to display.

See the Javadoc documentation for ResourceBundle.getString() at http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html for details.

The <blox:messageArg> Tag

The tag specifies the string values to be used as replacement values in the message resource using the java.text.MessageFormat syntax. This tag must be nested within a `<blox:message>` tag. It has one attribute:

Argument	Required	Description
`value`	Yes	The String value to be used as the replacement value in the message. See the Javadoc documentation for MessageFormat for details.

The `<blox:messageArg>` tag is an empty tag and therefore should be added using the shorthand notation:

```
<blox:message key="messagekey3">
   <blox:messageArg value="<%=argValue %>"/>
</blox:message>
```

Depending on the application server, you could get an error if you close it using a `</blox:messageArg>` closing tag.

Exceptions

The Blox Java APIs throw exceptions if they reach error conditions, and, if you want to, you can catch those exceptions and do something with them. For example, you might want to catch an exception and send a specific error message to the user with instructions how to continue. The exceptions thrown by each API are documented in the API signature in the syntax descriptions.

If you want to use some of the exceptions, they are documented in the Javadoc documentation shipped with DB2 Alphablox. The Javadoc documentation is available in the following locations:

- The DB2 Alphablox information center
- The DB2 Alphablox CD, under *documentation/javadoc*
- Through the Help link on the DB2 Alphablox Admin Pages

If you install the documentation locally on a Windows system, Windows shortcuts are created under the IBM DB2 Alphablox Online Documentation program group.

The general practice for catching exceptions is to use the `try...catch` syntax similar to the following pseudocode:

```
<%
try {
%>

<% original JSP Code %>

<% catch {Exception e}
{
   out.println(e.getMessage());
}
%>
```

Whether or not you catch exceptions, it is a good practice to add a custom error page. For details, see a JSP or Java reference book or the "Error Handling" section of the *Developer's Guide*.

Chapter 4. Common Blox Tag Reference

This chapter contains reference material for tag attributes and properties that are common to multiple Blox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

Common Blox Tag Attributes by Category

The following tables list the tag attributes and properties common to multiple Blox. The attributes and properties are organized as follows:

Application and Session Related Tag Attributes

The following tag attribute affects application instantiation and the user session.

- `helpTargetFrame`

Behavior and Appearance Related Tag Attributes

The tag attributes in this section affect Blox behavior and appearance.

- `bloxEnabled`
- `bloxName`
- `applyPropertiesAfterBookmark`
- `bookmarkFilter`
- `maximumUndoSteps`

Bookmark and Application State Related Tag Attributes

The following table lists tag attributes associated with bookmarks.

- `bookmarkFilter`
- `applyPropertiesAfterBookmark`

Rendering Related Tag Attributes

These tag attributes affect the delivery of Blox.

- `removeAction`
- `render`
- `rightClickMenuEnabled`

Menubar Related Tag Attribute

The tag attribute below determines if the menu bar is visible in a PresentBlox, GridBlox, or ChartBlox:

- `menubarVisible`

Popped Out Related Tag Attributes

The following table lists the tag attributes regarding displaying a PresentBlox, a standalone GridBlox, or a standalone ChartBlox in a separate, popped out browser window.

- `enablePoppedOut`
- `poppedOut`
- `poppedOutHeight`
- `poppedOutTitle`
- `poppedOutWidth`

Tag Attributes Common to Multiple Blox

This section describes the tag attributes supported by multiple Blox. The attributes are listed alphabetically. Each tag attribute typically maps to a Blox property with associated setter and getter Java methods. For methods common to multiple Blox, see com.alphablox.blox.Blox, com.alphablox.blox.ViewBlox and com.alphablox.blox.DataViewBlox in the Javadoc documentation.

applyPropertiesAfterBookmark

Specifies whether, after retrieving a bookmark, the Blox properties should override those in the bookmark.

Data Sources

All

Syntax

`applyPropertiesAfterBookmark="`*`applyAfterBookmark`*`"`

Argument	Default	Description
`applyAfterBookmark`	`false`	Specify `true` to apply any properties specified in the Blox custom tags after a bookmark is loaded.

Usage

A value of `true` overwrites the bookmark property values with those on the application page.

Note: The DataBlox `dataSourceName` property ignores the `applyPropertiesAfterBookmark` setting. If data source A is currently used by a PresentBlox on a page and the user loads a bookmark that was saved on data source B, data source B will be used and loaded even if `applyPropertiesAfterBookmark` is set to `true`.

Examples

```
applyPropertiesAfterBookmark="true"
```

bookmarkFilter

Specifies a default location from which to store and load bookmarks. You can use this location to provide grouping and visibility for bookmarks, providing application developers control over the set of bookmarks available to end users. The filter you specify causes the bookmarks to be stored in a subdirectory called `filterName` under the usual bookmark directory (`public`, `private`, or `group`) in the DB2 Alphablox repository.

Additionally, the `bookmarkFilter` property can allow bookmarks to be shared across multiple Blox and/or multiple applications.

Data Sources

All

Syntax

`bookmarkFilter="`*`filterName`*`"`

Argument	Default	Description
`filterName`	none	There are two forms of this argument. The first is a String specifying the name of subdirectory. The second form is a String with a comma separated list of the form: `"`*`subDirectory`*`, name=`*`BloxID`*`, application=`*`AppContext`*`, user=`*`UserName`*`"` The `name`, `application`, and `user` clauses are all optional; the *subDirectory* clause is required.

Usage

The `bookmarkFilter` property allows you to categorize the bookmarks of each Blox, providing greater flexibility of bookmarks.

For example, consider an application with a single PresentBlox that has two entry points (for example, two links that run different queries from different data sources, but display the results in the same PresentBlox), one for `marketing` and one for `sales`. You might want to set the `bookmarkFilter` property for the first link so that all users in `marketing` see bookmarks created from the marketing part of the application, and set the `bookmarkFilter` property for the second link so all users in `sales` see the bookmarks created in the sales part of the application. The net result is that users in `sales` see one set of bookmarks while users in `marketing` see another, even if they are both using the same PresentBlox to display the data. This scheme allows you to minimize the number of Blox on your page. Minimizing the number of Blox is a useful resource optimization tool, especially useful in applications running in Netscape browsers, where multiple page refreshes with multiple applets on the page sometimes causes the browser to behave unexpectedly.

The second form of the `filterName` argument described in the syntax section is useful in applications where the Blox is created dynamically based on programmatic details (for example, user profiles, fiscal quarter, etc.). Such dynamic Blox creation often results in the Blox having a different name each time the Blox is created, making it difficult to have a consistent set of bookmarks across different

Blox instances. Setting the bookmarkFilter based on some criteria (BloxID, application, and/or user) ensures the desired set of bookmarks are available to the appropriate users.

Examples

The following example sets the bookmarks so they are stored and retrieved in the marketingBookmarks subdirectory of the marketing application of the DB2 Alphablox repository.

```
bookmarkFilter="marketingBookmarks, name="myPresentBlox", application=marketing"
```

bloxEnabled

Specifies whether or not the Blox interface is interactive and greyed out.

Data Sources

All

Syntax

```
bloxEnabled="enable"
```

Argument	Default	Description
enable	true	Specify true to enable interactivity and display the Blox, false to disable interactivity and grey out the Blox.

Usage

A value of false presents a greyed-out Blox that is not interactive. A value of true presents an interactive interface, allowing the user to drill up and down, change chart types, and so forth. If you want the Blox to display (not greyed out), but do not want users to interact with the data, use the bloxui:component tag's clickable attribute. See "Example 3: Setting a PresentBlox Unclickable" on page 372.

Examples

```
bloxEnabled="false"
```

bloxName

Specifies the name of the Blox. This is an optional attribute and is an advanced feature that allows you to dynamically set the name of Blox and its corresponding JavaScript name.

Data Sources

All

Syntax

```
bloxName="bloxName"
```

Usage

When you define a Blox using Blox tags, you need to uniquely identify the outmost Blox using the id attribute (nested Blox cannot have a separate id). This id is required, cannot be dynamically set, and is used for two purposes:

- as the scripting variable name within your JSP page, and
- as the name of the Blox and its corresponding JavaScript object created under DB2 Alphablox (used in JavaScripting your Blox within the browser)

If the optional bloxName attribute is not specified, the id is used as both a Java scripting variable and the name of the Blox object on the server.

The use of id alone as both the Blox name and the Java scripting variable name is probably sufficient for all your development needs. Only in a few cases will you want to separate the two so you can dynamically create Blox names with tags. It is also useful for reusing server-side code (such as using the client-side call() method to execute server-side code that acts on the Blox or as demonstrated in the examples below). If you specify the value of bloxName for a Blox, then:

- this bloxName will be the name of the Blox DB2 Alphablox knows this object as
- this bloxName will be the name of the rendered JavaScript object (to be used in your JavaScript code when referencing the Blox)
- id will now only serve as the Java scripting variable you use in your JSP page.

Separating the scripting variable name and the Blox name: This following example demonstrates the differences between the scripting variable name and the Blox name when bloxName is specified. The code creates a GridBlox called salesDataGrid, which will not be displayed initially (visible="false"):

```
...
<blox:grid id="myGrid" bloxName="salesDataGrid"
    visible="false"
    width="400
    height="360"
    <blox:data dataSourceName="qcc"
        query="!" />
    <%
    //you can set properties using id within the grid tag as
    //it is now a scripting variable
    myGrid.setBandingEnabled(true);
    ....
    %>

</blox:grid>
...
//In your scriptlet within the page, you script to the grid using
//its id
<%
    myGrid.getDataBlox().setQuery(newQuery);
    myGrid.getDataBlox().connect();
%>
```

Notice that you script to this grid using its scripting variable name, which is the value of id. After some processing logic is done, this GridBlox is displayed using the <blox:display> tag:

```
//In other Blox tags that reference this Blox, use the Blox name
<blox:display bloxRef="salesDataGrid" />
```

Dynamically setting the value of the bloxName attribute: In this example, you have a JSP page named setAlerts.jsp that sets cell alert format for the specified threshold on any GridBlox that is passed in:

```
<!--This page is called by another JSP, with two parameters-->
<!--"blox" and "low" passed in along with the request.-->
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    //Blox name is passed in as a request parameter
    String gridName = request.getParameter("blox");
    String lowValue = request.getParameter("low");
%>

<blox:grid id="someGrid" bloxName="<%= gridName %>"/>

<%
```

```
    someGrid.setCellAlert(1,"condition=LT,value=" + lowValue +
",foreground=white,background=red");
    return;
%>
```

Or you want to reuse some generic JSP code:

```
<blox:grid id="someGrid" bloxName="<%=currBloxName %>"
    .... />
<%@ include file="gridDefaults.jsp" %>
```

where `gridDefaults.jsp` does the following:

```
someGrid.setBandingEnabled(true);
someGrid.setCellFormat(1, "format=#,##0.00,
      scope={Accounts:COGS}");
```

In summary:

- `id` is required; `bloxName` is optional.
- The name of a Blox is the value of the `bloxName` attribute if it is specified in the tag. If it is not supplied, the value of the `id` attribute is used both as the Blox name and the Java scripting variable name.
- Throughout this documentation set, the phrase "Blox name" refers to the value of the `id` attribute unless the value of the `bloxName` attribute is supplied.
- In most cases, you only need to specify the `id` and do not need to worry about `bloxName`. Only when you need to separate the scripting variable name and the Blox name, do you need to specify `bloxName`.
- If you do specify the value of the `bloxName` attribute:
  - `id` is the Java scripting variable name to script to in your JSP page
  - `bloxName` is the Blox name you should use when referencing it

Note: `bloxName` cannot be a number, start with a number, or contain any special characters such as `~`, `!`, `@`, `#`, `$`, `%`, `^`, `&`, `*`, `-`, `+`, `=`, `(`, `)`, `?`, `<`, `>`, `/`, `:`, `;`, `'`, or `"`.

Note: If you call the `getBloxName()` method on a nested Blox, it returns the name generated for the Blox.

Examples

The following code creates a local scripting variable named `myGrid` and a Grid peer named `salesGrid`.

```
<% String bloxName="salesGrid"; %>
<blox:grid id="myGrid" bloxName="<%= bloxName %>" .../>
```

To script to this grid, use the grid's scripting variable name (`id`). The following code show the results of the `getBloxName()` method. The comments indicate the returned values.

```
<%
    myGrid.getBloxName(); // returns the string "salesGrid"
    myGrid.getDataBlox().getBloxName();
    //returns the generated name for the nested DataBlox (for
    //example, "salesGrid_data")
```

See Also

"id" on page 36

bloxRef

Specifies the name of another Blox to use. The `bloxRef` attribute is available through the DataBlox (`blox:data`) and display (`blox:display`) custom tag libraries.

Data Sources
All

Syntax
`bloxRef="`*`bloxName`*`"`

Usage
Use the `bloxRef` tag attribute in a nested Blox to refer to a Blox that was created as a separate Blox.

Examples
If the following DataBlox was created in the `<head>` section of an HTML page:

```
<blox:data id="DataBlox1"
           dataSourceName="TBC"
           query="!"
</blox:data>
```

You could then reference that DataBlox within other Blox (for example, GridBlox) as a nested Blox, referring to it with the `bloxRef` attribute as follows:

```
<blox:grid id="myGrid" >
    <blox:data bloxRef="DataBlox1" />
</blox:grid>
```

enablePoppedOut

This is a property inherited from ContainerBlox. For a complete description, see "enablePoppedOut" on page 152.

height

Specifies the height of the Blox on the page.

Data Sources
All

Syntax
`height="`*`height`*`"`

Usage
Specifies the height of the Blox display area. The value can be expressed as pixels (`height="300"`) or as a percentage of the browser display area (`height="40%"`).

helpTargetFrame

Identifies the target browser window or frameset frame in which user help appears.

Data Sources
All

Syntax
`helpTargetFrame="`*`helpTargetFrame`*`"`

Argument	Default	Description
helpTargetFrame	"AlphabloxHelp"	String identifying a browser window or frameset.

Usage
The default is AlphabloxHelp, which is a separate browser window.

Examples
```
helpTargetFrame ="Browser Window Name"
```

id

Specifies the name of the Blox. This name can then be referenced from other Blox or from Java or JavaScript code on the JSP page.

Data Sources
All

Syntax
```
id="idString"
```

Argument	Default	Description
idString	none	An identification string representing the name of the Blox. The idString can be referenced from other Blox tags using the bloxRef attribute.

Usage
The id attribute is valid only on the outer Blox; nested Blox cannot have an id attribute. If you specify the optional bloxName attribute, then id will serve only as the Java scripting variable name in your JSP page. The value of bloxName will be the name of the Blox peer created on the server and the name of the rendered JavaScript object.

Note: id cannot be a number, start with a number, or contain any special characters such as ~, !, @, #, $, %, ^, &, *, -, +, =, (,), ?, <, >, /, :, ;, ', or ".

See Also
"bloxName" on page 32

maximumUndoSteps

Specifies the maximum number of steps to be tracked in the menu bar's Undo button.

Data Sources
All

Syntax
```
maximumUndoSteps="steps"
```

Argument	Default	Description
steps	50	The maximum numbers of steps to be tracked.

Usage
This property applies to PresentBlox, GridBlox, ChartBlox, DataLayoutBlox, and PageBlox. When this property is set to 0, the Undo and Redo buttons and menu items will be removed from the toolbar and menu bar.

menubarVisible
Specifies whether or not a text-based menu bar will appear at the top of the Blox.

Data Sources
All

Syntax
`menubarVisible="visible"`

Argument	Default	Description
`visible`	`true`	Set to `true` to show the menu bar, set to `false` to hide the menu bar. The default is `true` (when the default application render mode is set to DHTML).

Usage
The contents of the menu bar and its drop-down menus automatically match the contents of the Blox.

Examples
```
menubarVisible="true"
```

noDataMessage
Sets the string to be displayed on the Blox when it has no data.

Data Sources
Multidimensional

Syntax
`noDataMessage="message"`

Argument	Default	Description
`message`	`"Data not available"`	Any string.

Usage
When a new message is set, the screen will not get updated until next time a result set is returned with no data (such as after a call to `updateResultSet()` or `connect()`). With DataLayoutBlox, this property is ignored and no message appears if no data is available.

Examples
```
noDataMessage="No data at this time"
```

poppedOut
This is a property inherited from ContainerBlox. For a complete description, see "poppedOut" on page 153.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 154.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 154.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 155.

removeAction

Specifies which (if any) data analysis actions to remove from the right-click menu and the Data menu in the menu bar.

Data Sources

All

Syntax

```
removeAction="dataActions"
```

Argument	Default	Description
dataActions	empty string	Comma-delimited list of actions to remove.

Usage

Valid entries in the list are:

- Find
- Drill Up
- Drill Down
- Pivot
- Data Sort
- Remove Only
- Keep Only
- Hide Only
- Show Only
- Show All
- Expand All
- Show Bottom Level
- Show Siblings
- Swap
- Drill Through
- Member Filter
- Comments
- Traffic Lights

Examples

```
removeAction="Keep Only, Remove Only, Pivot"
```

render

Specifies the delivery format for a specific Blox on an application page.

Data Sources
All

Syntax
render="*renderMode*"

Argument	Default	Description
renderMode	dhtml	The mode in which the page is rendered. See the table below for the possible values.

Usage
Using this property enables different delivery formats for Blox on the same page. Setting this property on an individual Blox overrides the render attribute on the application's URL. The render attribute applies to all Blox on the page; this property applies to a specific Blox. Therefore, to ensure that a Blox is delivered in a specific format only, use the render property on the Blox.

Possible values are:

Value	
dhtml	Render in fully interactive DHTML format (the default). Requires the <blox:header> tag in your JSP page
html	Render in static HTML format. Requires the <blox:header> tag in your JSP page.
printer	Render in a format suitable for printing (many browsers do not support printing the output of interactive Java applets). Requires the <blox:header> tag in your JSP page.
xls	Render a format suitable for export to Microsoft Excel, and set the MIME type to XLS. **Note:** In order for the MIME type to be set so the page opens in Excel, you must put the <blox:header> tag in your JSP page. For information on the <blox:header> tag, see "<blox:header> Tag in the HTML <head>" on page 18.
xml	Render in XML format. This format applies only to DataBlox. For more information, see Appendix B, "Using the Alphablox XML Cube," on page 481. **Note:** To render to Microsoft Excel format, you must use the URL render=xls attribute.

rightClickMenuEnabled
Specifies whether the right-click menu in the Blox user interface should be turned on or off.

Data Sources
All

Syntax

`rightClickMenuEnabled="enabled"`

Argument	Default	Description
`enabled`	`true`	When set to `true`, the right-click menu is enabled, allowing users to perform data analysis or manipulation tasks. When set to `false`, the right-click menu is disabled. The default is `true`.

Usage

This property only has an effect on the right-click menu in GridBlox and ChartBlox, either standalone, or nested inside a PresentBlox.

visible

Specifies whether a Blox is visible on the page.

Data Sources

All

Syntax

`visible="boolean"`

Argument	Default	Description
`boolean`	`true`	Set to `true` to cause the Blox to render to the page, set to `false` if you want the Blox created but not to display on the page.

Usage

Set the visible property to `false` to create a Blox but not display it. You can later display the Blox using the `<blox:display>` tag. The default value is `true`.

If using the `visible` property on ToolbarBlox, carefully consider the user implications of turning the toolbar off. Most applications need to provide some of the functionality provided through either the Blox toolbars or menu bars. If the menu bar is turned off in a Blox, options such as Undo/Redo buttons, export to PDF/Excel, and turning on/off the grid, the chart, the page filter, and the data layout panel are only available through the toolbars.

width

Specifies the width of the Blox on the page.

Data Sources

All

Syntax

`width="width"`

Argument	Default	Description
`width`	none	A string representing the valid HTML width values.

Usage

Specifies the width of the Blox display area. The value can be expressed as pixels (`width="500"`) or as a percentage of the browser display area (`width="50%"`).

Chapter 5. AdminBlox Tag Reference

This chapter contains reference material for AdminBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

AdminBlox Overview

AdminBlox provides programmatic access to information on the server, users, groups, roles, data sources, the Alphablox log system, and applications set through the Administration tab in the DB2 Alphablox Home Page.

The Administration tab under the DB2 Alphablox Home Page provides a means for server administrators to specify properties such as server log file name, message level, telnet console port, clustering options, and telnet username and password. The Data Sources, Users, Groups, Roles, Cubes, and Applications links under the Administration tab allow administrators to define data sources, users, groups, roles, cubes, and applications to be served by DB2 Alphablox. Once specified, this information is stored in the repository. Application developers can access this information via AdminBlox and its related objects and methods.

	Method		Object Returned
	`getApplication(...)` `getApplications()`	—>	Application
	`getCube(...)` `getCubes()`	—>	Cube
	`getDataSource(...)` `getDataSources()`	—>	DataSource
AdminBlox —>	`getGroup(...)` `getGroups()`	—>	Group
	`getLog(...)`	—>	Log
	`getRole(...)` `getRoles()`	—>	Role
	`getServer(...)`	—>	Server
	`getUser(...)` `getUsers()`	—>	User

While AdminBlox and RepositoryBlox both let you access information stored in the repository, AdminBlox is specific to the general, administrative details on the server, applications, users, and data sources. For example, it allows you to get information regarding all data sources available to the server, find out the session timeout setting of an application, or identify the specified SMTP server. RepositoryBlox, on the other hand, gives you access to information on the current user, application, and custom properties.

With AdminBlox, you can build your own administration application to serve your specific server monitoring and management needs. Because of this power, care should be taken at the application level to ensure the appropriate access control is in place. There is no built-in security for AdminBlox.

The Application Object

The Application object represents an Alphablox application in the repository. It provides methods for you to get information specified through the Application link under the Administration tab. With the getter methods provided, you can find out information such as an application's default saved state, display name, header links defined, and the amount of inactive time for the session to time out.

The Cube Object

The Cube object represents an Alphablox cube in the repository. It provides methods for you to access a cube's definition specified through the Cubes link under the Administration tab. It also provides methods for managing the cube, such as `start()`, `stop()`, and `refreshData()`. Once you get a cube's definition (the CubeDefinition object), you can access the high level metadata about the cube, such as dimensions, levels, calculated and default members of dimensions. It also contains some administrative information about the cube, such as refresh intervals and cube cache seeding queries upon cube startup.

The DataSource Object

The DataSource object represents an Alphablox data source in the repository. It provides methods for you to get information specified through the Data Source link under the Administration tab. With the getter methods provided, you can find out information such as the data source's adapter type, application/catalog/database/schema name, default username and password to log in to the data source, and maximum columns and rows to return.

The User Object

The User object represents an Alphablox user in the repository. It provides methods for you to access information specified through the Users link under the Administration tab. With the getter and setter methods provided, you can find out and modify information such as the user's name, email address, and primary group association.

The Group Object

The Group object represents an Alphablox group in the repository. It provides methods for you to access information specified through the Groups link under the Administration tab. With the getter and setter methods provided, you can find out and modify information such as which users or subgroups belong to a specified group.

The Role Object

The Role object represents an Alphablox role in the repository. It provides methods for you to access information specified through the Roles link under the Administration tab. With the methods provided, you can find out and modify information such as which users or groups belong to a specified role.

The Log Object

The Log object is used to place messages into the Alphablox log system. The message levels are, in order of severity from minor to the most severe: DEBUG, VERBOSE, INFO, SYSTEM, WARNING, and ERROR. These messages will be logged to the log file and registered consoles depending on their message level setting. The log file is located inside the DB2 Alphablox repository under the `<alphablox_dir>/repository/logs/<instance_name>/logs`.

The Server Object

The Server object represents server-related information that DB2 Alphablox stored in the repository. It provides methods for you to get information specified through the Server link under the Administration tab.

Server Message Level

The DB2 Alphablox provides seven message levels to log messages for server monitoring and debugging purposes. Administrators can specify the New Log Start Message Level and New Log End Message Level in the System page under the Administration tab. Setting the values on these two properties creates a log containing messages within the range of the specified levels.

The following table lists the message level constants and a description for the kind of message each level logs. It also shows the string and integer values used by the `levelIntToString()` and `levelStringToInt()` methods of the Server object in the com.alphablox.blox.repository package. To get to the Server object, use the `AdminBlox.getServer()` method.

Message Level Constant	String	Value	Description
`MESSAGE_LEVEL_DEBUG`	`DEBUG`	1	Messages that aid debugging the system.
`MESSAGE_LEVEL_VERBOSE`	`VERBOSE`	2	All system messages.
`MESSAGE_LEVEL_INFO`	`INFO`	3	Minor system events for which no administrator action is necessary.
`MESSAGE_LEVEL_SYSTEM`	`SYSTEM`	4	Messages for major, normal system events (such as new sessions).
`MESSAGE_LEVEL_WARNING`	`WARNING`	5	Messages that indicate a minor, recoverable, error has occurred, suggesting issues the administrator may want to investigate.
`MESSAGE_LEVEL_ERROR`	`ERROR`	6	Messages that indicate an operation cannot be completed and a non-recoverable error has occurred.
`MESSAGE_LEVEL_FATAL`	`FATAL`	7	Messages that indicate that a fatal error has occurred that can cause the server to terminate.

AdminBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create an AdminBlox. For a copy and paste version of the tag with all the attributes, see "AdminBlox JSP Custom Tag" on page 457.

Syntax

```
<blox:admin
    [attribute="value"] >
</blox:admin>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
"id" on page 45
"bloxName" on page 45

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing </blox:admin> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myAdminBlox" />
```

Examples

```
<blox:admin
     id="namedAdminBlox" />
```

AdminBlox Example

This example demonstrates how to log messages to the Alphablox log system through AdminBlox. This is particularly useful in monitoring, logging, and debugging problems. It gives you the ability to log both messages and Exceptions.

Note the logging mechanism is multi-threaded so that you could get messages slightly out of the order from what you expect.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ page import="com.alphablox.blox.repository.Log" %>
<html>
<head>
   <blox:header />
</head>

<body>
<blox:admin id="myAdminBlox" />
<%
   Log log = myAdminBlox.getLog();
   log.sendMessage( Log.MESSAGE_LEVEL_INFO, "My Info Message Title",
"My Info Message" );
   Exception e = new Exception( "My dummy Exception" );
   log.sendException( Log.MESSAGE_LEVEL_INFO, "My Info Exception
Title", e);

%>
The Log test is done.
</body>
</html>
```

1. Import the com.alphablox.blox.repository.Log class.
2. Add an AdminBlox using the <blox:admin> tag.
3. Access the Log object via AdminBlox's getLog() method.
4. Send a message to the log using sendMessage(...).
5. Send an Exception to the log using sendException(...).

This would generate the following entries in the log file:

7/28/04 1:29:52 PM [INFO] My Info Message Title: My Info Message 7/28/04 1:29:52 PM [INFO] My Info Exception Title: My dummy Exception

7/28/04 1:29:52 PM [INFO] My Info Message Title: My Info Message 7/28/04 1:29:52 PM [INFO] My Info Exception Title: My dummy Exception

```
java.lang.Exception: My dummy Exception
at org.apache.jsp._log4._jspService(_log4.java:126)
at com.ibm.ws.webcontainer.jsp.runtime.HttpJspBase.service(HttpJspBase.java:89)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
at
com.ibm.ws.webcontainer.jsp.servlet.JspServlet$JspServletWrapper.service(JspServlet.ja
va:344)
at com.ibm.ws.webcontainer.jsp.servlet.JspServlet.serviceJspFile(JspServlet.java:662)
at com.ibm.ws.webcontainer.jsp.servlet.JspServlet.service(JspServlet.java:760)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:853)
at
com.ibm.ws.webcontainer.servlet.StrictServletInstance.doService(StrictServletInstance.ja
va:110)
at
com.ibm.ws.webcontainer.servlet.StrictLifecycleServlet._service(StrictLifecycleServlet.ja
va:174)
[ more stack traces below omitted... ]
```

AdminBlox Tag Attributes

This section describes the tag attributes supported by AdminBlox. The use of AdminBlox to monitor and manage server resources requires the use of AdminBlox API. For details on AdminBlox methods, see com.alphablox.blox.AdminBlox in the Javadoc documentation.

id

This is a common Blox tag attribute and property. For a complete description, see "id" on page 36.

bloxName

This is a common Blox tag attribute and property. For a complete description, see "bloxName" on page 32.

Chapter 6. BookmarksBlox Tag Reference

This chapter contains a general overview of bookmarks and reference material for BookmarksBlox tag attributes. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

BookmarksBlox Overview

Through the Blox user interface, end-users can bookmark data views with either private, public, or group accessibility for later retrieval. Bookmarking a view is done via the Bookmark button in the Toolbar or the Bookmark option from the right-click menu. Users can also load, delete, or rename existing bookmarks that are visible to them.

A bookmark is essentially a collection of property sets. Each bookmark contains the following information:

- the name of the Blox whose state is stored
- the change in properties from the initial application state of the Blox to the current state when the bookmark is added
- the name of the user who owns the bookmark
- the bookmark's visibility
- some description about the bookmark

When a bookmark is saved, only the difference between a Blox's current state (after the user interacts with the data) and its initial state (the default property values or the values specified when the Blox is created) are stored in the repository. When a bookmark is loaded, live data is retrieved from the data source based on the Blox properties information stored in the repository.

BookmarksBlox, with its extensive API, allows you to programmatically create and manage bookmarks and dynamically set the bookmark properties. For example, you can create time-series reports or reports that always fetch the data for the current quarter by dynamically modifying the data query stored with a bookmark. You can use custom bookmark properties to store each user's choice of report layout or implement your own security. You can modify the query stored with a bookmark in the case of change of member names or outline in the data source. You can even create your own bookmark management user interface.

To use the BookmarksBlox API, add a BookmarksBlox to your page. This gives you access to each bookmark as a Bookmark object.

Bookmark Concepts and Features

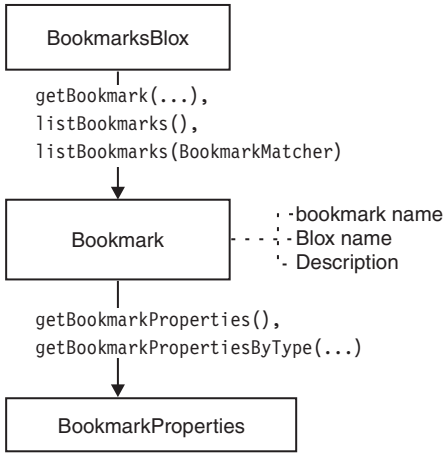
Bookmarking is a powerful feature with an extensive API that allows you to perform various custom actions. This section discusses the following key concepts and features of bookmarks and related Bookmark objects:

What is a Bookmark?

A Bookmark is a collection of property sets. A Bookmark, by itself, has its own properties like application, description, name, and visibility. It also stores information with regard to an individual Blox. This Blox can be a standalone Blox that has no nested Blox (such as a DataBlox), or a Blox with nested Blox (such as a PresentBlox). For example, if a Bookmark is added on a PresentBlox, information on the individual nested Blox is also stored.

You can use the BookmarksBlox API to access specific bookmarks by specifying the search criteria such as the bookmark's name, the Blox name, the owner's name, and its visibility. In addition, the BookmarksBlox API allows you to modify the properties or even apply the bookmark to a different Blox.

The following diagram shows the object hierarchy of BookmarksBlox.



Note: To access the Bookmark and BookmarkProperties objects, you should add the following import directive in your JSP:

```
<%@ page import="com.alphablox.blox.repository.*" %>
```

Note: Bookmark names can only contain the following characters: A-Z, a-z, 0-9, and underscore (_).

Blox Default States vs. Initial Application State vs. Current Blox State

The `BookmarkProperties` object only contains properties that are not the same as the initial Blox state. For example, if ChartBlox's `chartType` is not set in the tags, the default chart type is "Vertical Bar, Side-by-Side, 3D Effect." If a bookmark is saved and the current displayed chart type is "Vertical Bar, Side-by-Side, 3D Effect," then the chart type property will not be in the list of properties stored for the Blox.

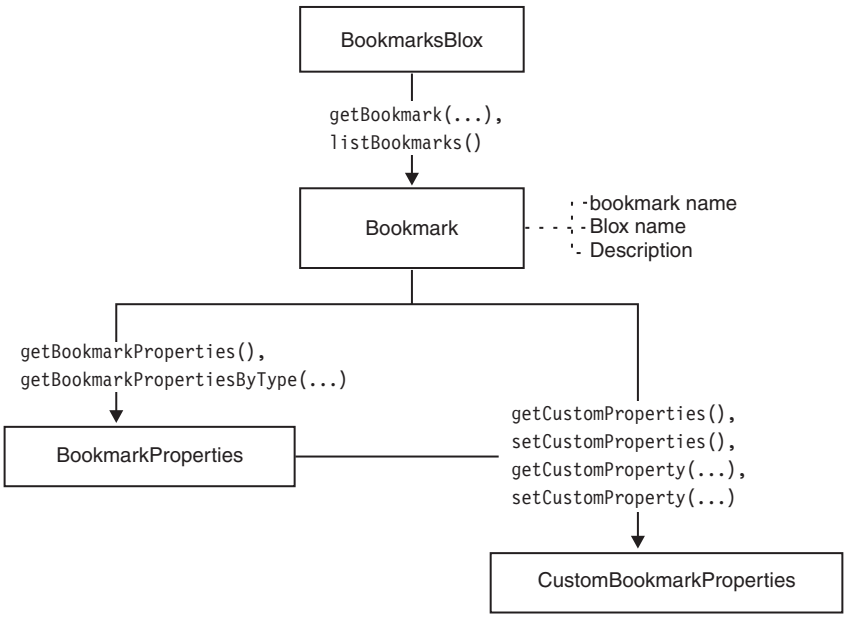
Besides Blox default states, you might set the chart type to "Pie" through the ChartBlox tag. This specified property, together with the default values of the other unspecified Blox properties, dictates how the Blox are instantiated and rendered. This is the initial application state. The state is changed when a user interacts with the data, such as changing the chart type, drilling down, hiding members, swapping axes, or setting some other cell banding style. When he saves a bookmark on the current view, what is stored with the bookmark is the difference between the initial application state and the current Blox state.

Custom Bookmark Properties

Besides the default bookmark properties, you can also add custom properties to bookmarks. Similar to the custom user properties and custom application properties available in RepositoryBlox, custom bookmark properties allow you to store any information in a name-value pair that you need in your application. For instance, you might want to build a navigation tree menu for bookmarks. Using the custom bookmark properties, you can store the folder names for dynamically building the tree menu. Or you can implement access control so only certain users or groups can see certain folders in your navigation tree. These properties do not affect the behavior of bookmarks in any way but allow you to save and get custom properties.

Custom bookmark properties are different from custom user/application properties in that they are not defined through the DB2 Alphablox Admin Pages and are not accessed via the RepositoryBlox. To create and access custom bookmark properties, first add a BookmarksBlox to your JSP file and then you can:

- Use the `BookmarksBlox.getBookmark(...).setCustomProperties()` method to set custom properties
- Use the `BookmarksBlox.getBookmark(...).getCustomProperties()` method to get all custom properties associated with a bookmark
- Access individual custom properties by their key using the `getCustomProperty(`*key*`)` method.



Bookmark Visibility

A bookmark can be private, public, or group visible only. By default, bookmarks are added as private bookmarks unless the users (through the Blox user interface) or developers (through BookmarksBlox API) specify otherwise. Bookmark visibility is marked using the following static fields:

- `PRIVATE_VISIBILITY`
- `PUBLIC_VISIBILITY`

For group visibility, use the name of the group to get group bookmarks.

Blox Types and Binding

When a bookmark is saved on a Blox through the Blox user interface, properties of all nested Blox are also saved if they are not in their initial state. If a bookmark on a PresentBlox is saved, in the folder for the bookmark, a separate folder is created for each of the nested Blox if the Blox is in a state different from its initial state:

- `<blox name>_data` (if using an implicit DataBlox that is not explicitly defined with an `id` outside a presentation Blox)
- `<blox name>` (for the PresentBlox)
- `<blox name>_chart`
- `<blox name>_datalayout`
- `<blox name>_grid`
- `<blox name>_page`
- `<blox name>_toolbar`

Through the BookmarksBlox API, you can access the property set of a nested Blox by specifying its Blox type. Blox types are marked using static fields:

- `CHART_BLOX_TYPE`
- `DATA_BLOX_TYPE`
- `DATALAYOUT_BLOX_TYPE`

- GRID_BLOX_TYPE
- PAGE_BLOX_TYPE
- PRESENT_BLOX_TYPE
- TOOLBAR_BLOX_TYPE

This allows you to directly access and modify the property set of a specific Blox type.

The physical location of the bookmark is called the binding. A binding is the association of an object with a logical name and a context. It is based on the Java Naming and Directory Interface (JNDI), which provides Java technology-enabled applications with a unified interface to seamlessly navigate across databases, files, directories, objects, and networks. J2EE containers use this information to locate needed resources. Using the `getContainer()` and `getBinding()` methods on the Bookmark object, you can get the physical location of a bookmark.

Bookmark Matchers and Bookmark Filters

You can find a list of bookmarks that match a certain criterion, or get a list of bookmarks for an application, for a specific group of users, or for a specific user. Since application, user, and group specific information is stored in the repository, objects supporting bookmark filtering are in the `com.alphablox.blox.repository` package. These objects include BookmarkMatcherApplications, BookmarkMatcherGroups, BookmarkMatcherUsers, and BookmarkMatcherAll.

The BookmarkMatcherApplications object is used to find bookmarks based upon which application owns a bookmark. An application bookmark is equivalent to a public bookmark. The BookmarkMatcherGroups object is used to find bookmarks based upon which group owns a bookmark. The BookmarkMatcherUsers object is used to find bookmarks based upon which user owns a bookmark. A user bookmark is equivalent to a private bookmark. The BookmarkMatcherAll object lets you find all bookmarks for a specified application, user, visibility or Blox name.

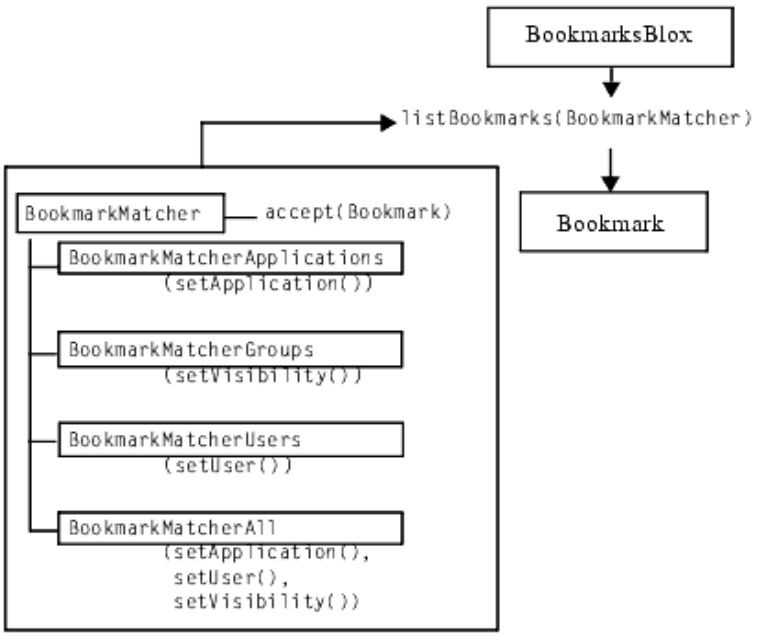
All these BookmarkMatcher objects work in much the same way as an extended Java SDK File Filter class except that BookmarkMatcherUsers has a `setUser()` method that can be optionally called to find specific bookmarks for a user; BookmarkMatcherApplications has a `setApplication()` method that can be optionally called to find specific bookmarks for an application; and BookmarkMatcherGroups has a `setVisibility()` method that can be optionally called to find specific bookmarks for a group. Each of these objects has an `accept()` method. This method is called for every Bookmark object to see if it should be included in the list of bookmarks returned.

You can use these objects or extend them depending on what type of bookmark matching you want to perform. It is recommended that if any type of custom application/group/user bookmark matching is to be performed that it uses or extends BookmarkMatcherApplications, BookmarkMatcherGroups, BookmarkMatcherUsers, or BookmarkMatcherAll. DB2 Alphablox has optimized these classes to perform quick searches for applications, groups, and users.

Note: To access all these BookmarkMatcher objects, you should add the following import directive in your JSP:

```
<%@ page import = "com.alphablox.blox.repository.*" %>
```

The following diagram shows the how these objects are related to the Bookmark object.



Bookmark Events and Event Filters

You can intercept events when a user clicks to delete, edit, add, or save a bookmark. Using the server-side event filters, you can intercept the events and perform some actions *before* the server processes them. To use server-side event filters, generally involves two steps.

1. First you add the specific event filter object using the common Blox method `addEventFilter()`. For example,

```
<blox:present id="myPresent">
...
<%
  myPresent.addEventFilter(new LoadFilter() );
%>
</blox:present>
```

2. Then write your own class that implements the corresponding event filter object (`BookmarkDeleteFilter`, `BookmarkLoadFilter`, `BookmarkRenameFilter` and `BookmarkSaveFilter`) and the corresponding method (`bookmarkDelete(BookmarkDeleteEvent)`, `bookmarkLoad(BookmarkLoadEvent)`, `bookmarkRename(BookmarkRenameEvent)`, or `bookmarkSave(BookmarkSaveEvent)`) that will be called with the event is triggered. For example:

```
public class LoadFilter implements BookmarkLoadFilter
{
   public void bookmarkLoad( BookmarkLoadEvent bre )
   {
      //actions to take when the event is triggered
   }
}
```

For more information on bookmark events and event filters, see “Example 5: Using server-side bookmarkLoad event filter” on page 60, “Bookmark and Application State Related Tag Attributes” on page 29.

Serialized Query and Textual Query

When a bookmark is first created, the *delta* between the original query set in the underlying DataBlox and the associated query that generates the current data view is saved as well. In the case of a file repository, two files are saved in the bookmark's `<blox name>_data` folder in the repository— *bookmarkName*`.data` and *bookmarkName*`.query`. The `.data` file is a text file that contains the basic properties for reconnecting to the data source, such as application name, data source name, last executed query, and the page axis members. It looks like the following:

```
Associated.query = q2report
ResultSet.Market = East,West,South,Central,Market
applicationName = SalesApp
connectOnStartup = true
dataSourceName = TBC
dimensionsOnPageAxis = {[null]}
parentFirst = {[null]}
query = <Row(Market) <ICHILD Market <Column(Year) Year !
```

Textual Queries

The `.data` text file is created when a bookmark is first added. Depending on how the bookmark is created, you may or may not see the `query` entry. If the bookmark is created through the API by setting the bookmark object's `query` property, you will see the query string in the text file. This file, however, is not updated when the bookmark is resaved. To keep the textual query in sync with the serialized query when users try to resave a bookmark with a different data view:

1. First use the DataBlox `generateQuery()` method to get the textual query for the current data view.
2. Use the `addEventFilter()` common Blox method to add a method that implements the `BookmarkSaveFilter` interface to update the query stored in the bookmark every time a bookmark is resaved.

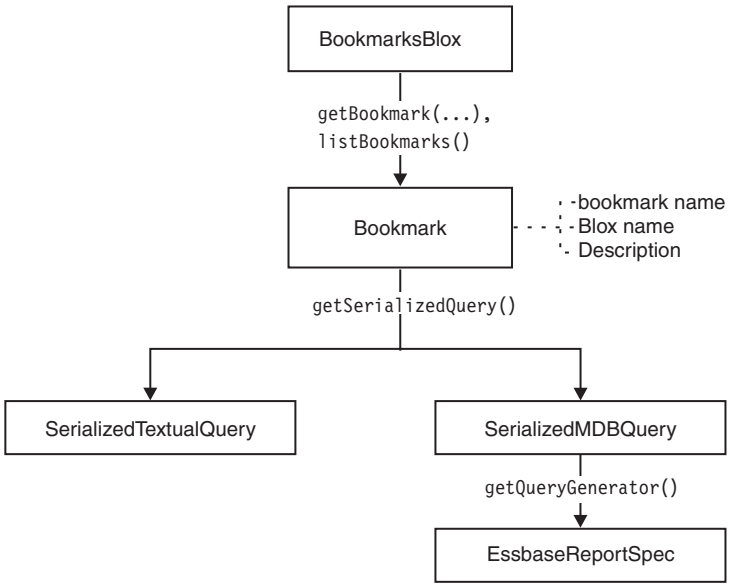
Keeping the textual query in sync allows you to modify the textual query later in cases such as data outline change. Textual queries can be more efficient since DB2 Alphablox does not need to manipulate the result set to match the serialized object.

When a bookmark is loaded, by default, the serialized query is used. To load a bookmark using the textual query, set the DataBlox `textualQueryEnabled` property to `true`. For an example of how to change the query before a bookmark is loaded, see "Example 6: Getting a bookmark's query when it is loaded" on page 61.

Serialized Queries

The `.query` file contains the serialized object that, in many ways, are similar to the GridBlox result set except that it has no data. It stores information on the axes, tuples, dimensions, and members. You can programmatically access the axes, tuples, dimensions, and members and modify the query before a bookmark is loaded. Or you can modify all bookmarks stored in the repository in cases where member names or the data outline have changed.

The following diagram shows how you can access the SerializedMDBQuery object (for multidimensional data sources) and the SerializedTextualQuery object (for relational data sources) through BookmarksBlox. The SerializedMDBQuery object lets you get information on the axis, dimension, tuple and member involved and replace a old member with a new member. You can also access the EssbaseReportSpec object in order to obtain specific Essbase Report Scripts. The SerializedTextualQuery object lets you get the saved query and set a new query.



Static Fields for the Bookmark Object

The Bookmark object contains the following static fields to indicate Blox type, bookmark visibility, and null dimension:

Static Field by Category

Blox Type

CHART_BLOX_TYPE

DATA_BLOX_TYPE

DATALAYOUT_BLOX_TYPE

GRID_BLOX_TYPE

PAGE_BLOX_TYPE

PRESENT_BLOX_TYPE

TOOLBAR_BLOX_TYPE

UNKNOWN_BLOX_TYPE

Bookmark Visibility

PRIVATE_VISIBILITY

PUBLIC_VISIBILITY

Null Dimension

NULL_DIMENSION

These static fields give you a way to specify and identify Blox type and bookmark visibility using constants.

Restrictions on Bookmark Names

There are several restrictions on the name of a bookmark:

- The name cannot be null or blank.
- The name cannot be any of the following reserved words, regardless of cases:
  - `properties`
  - `public`
  - `private`
- Allowable characters are A-Z, a-z, 0-9, dashes ("-"), underscores ("_"), and spaces.
- The name cannot contain leading or trailing spaces.

BookmarksBlox JSP Custom Tag Syntax

The Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to create the custom tag to create a BookmarksBlox. For a copy and paste version of the tag with all the attributes, see "BookmarksBlox JSP Custom Tag" on page 457.

Parameters

```
<blox:bookmarks
    [attribute="value"] >
</blox:>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Valid attributes are listed in the following table:

Attribute
id
bloxName

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:bookmarks>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myBookmarksBlox" />
```

Examples

```
<blox:bookmarks
   id = "myBookmarksBlox" />
```

BookmarksBlox Examples

This sections provide examples that demonstrate how to use BookmarksBlox, its associated objects and related methods:

Example 1: Getting a count of all bookmarks

This example demonstrates the following:

- the use of BookmarksBlox and its `listBookmarks()` method to gain access to all bookmarks stored in the repository. The `listBookmarks()` method returns an array of bookmark objects
- how to get a count of the total number of bookmarks by getting the length of the array

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<!--import the following package in order to access the
    com.alphablox.blox.repository.Bookmark class-->
<%@ page import="com.alphablox.blox.repository.*" %>

<blox:bookmarks id="myBookmarksBlox"/>

<%
   Bookmark bks[] = null;
   bks = myBookmarksBlox.listBookmarks();
%>
There are <%= bks.length %> bookmark(s).
```

Example 2: Getting the properties set for a Bookmark

This example demonstrates how to access a bookmark based on the bookmark name, application name, user name, Blox name, and bookmark visibility and get information on its properties set. In particularly, it demonstrates:

- the use of the BookmarksBlox to access individual bookmarks (the Bookmark object)
- the use of the Bookmark object's `getName()`, `getVisibility()`, `getDescription()`, `getBloxType()`, and `getBinding()` methods
- the use of the Bookmark object's `getBookmarkProperties()` method to access the individual properties (one for each nested Blox)

The generated output looks like the following:

The bookmark you are looking for exists.

1. The Repository JNDI binding for this bookmark is:
   `users/admin/salesapp/mygrid/bookmark/q2fy02WestSales/properties`
2. The bookmark name is: q2fy02WestSales
3. The type of Blox this bookmark was saved for is: grid
4. The bookmark description is: The Q2 West Sales
5. The bookmark visibility is: private

6. The bookmark contains Blox properties in the repository
7. Types of Blox properties saved in the bookmark:
   - grid
   - data

The code is as follows:

```
<%@ page import="com.alphablox.blox.repository.*,
                 com.alphablox.blox.ServerBloxMissingResourceException,
                 com.alphablox.blox.ServerBloxException,
                 com.alphablox.blox.BookmarksBlox" %>
<%@ page import="java.util.*" %>
<%@ page import="java.io.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<html>
<head>
<!-- Blox header tag -->
<blox:header/>
</head>

<body>
<!-- Get an SSPM BookmarkBlox -->
<blox:bookmarks id="bookmarks" />
<ol>

<!--getting the bookmark you want-->
<%
   String bookmarkName = "q2fy02WestSales";
   String applicationName = "SalesApp";
   String userName = "admin";
   String bloxName = "myGrid";
   String visibility = Bookmark.PRIVATE_VISIBILITY;
   Bookmark bk = null;
   try {
      bk = bookmarks.getBookmark(bookmarkName, applicationName, userName,
bloxName, visibility);
   } catch (ServerBloxException e){
      if (e instanceof ServerBloxMissingResourceException){
         %><li><b>The bookmark DOES NOT EXIST!</b></li><%
      }
      else
         throw e;
   }
   if (bk != null)  {
      %><li><b>The bookmark you are looking for exists.</b></li>
   <li><b>The Repository JNDI binding for this bookmark is:<br>
         <%= bk.getBinding() %></b></li>
   <li><b>The bookmark name is: <%= bk.getName() %></b></li>
   <li><b>The type of Blox this bookmark was saved for is: <%=
      bk.getBloxType() %></b></li>
   <li><b>The bookmark description is: <%= bk.getDescription() %>
      </b></li>
   <li><b>The bookmark visibility is: <%= bk.getVisibility() %>
      </b></li>

   <!---Getting the individual BookmarkProperties ---->
   <%
      BookmarkProperties props[] = bk.getBookmarkProperties();
      if (props != null) {
         %><li><b>The bookmark contains Blox properties in the
         repository</b><br>
         Types of Blox properties saved in the bookmark:
         <ul>
         <%
         for (int i = 0; i < props.length; i++) {
```

```
            %><li><%= props[i].getType() %></li><%
          }
          %></ul><br></li><%
       }
       else {
          %><li><b>The bookmark DOES NOT CONTAIN Blox properties in the
          repository</b></li><%
       }
    }%>
 </body>
 </html>
```

Example 3: Getting a list of bookmarks that match the specified criteria

This example demonstrates the following:

- getting bookmarks for a specified user, and in this example, the user "admin" with the use of the BookmarkMatcher object
- the use of the Bookmark object's `getBinding()` and `getBloxType()` methods and their output

The generated output is as follows:

Got 5 Bookmark Object(s) for user admin.

The Bookmarks are:

users/admin/salesapp/salesgrid/bookmark/salesq1fy03/properties (grid)

users/admin/salesapp/salespresent/bookmark/eastq2fy03/properties (present)

users/admin/budgetapp/mypresent/bookmark/eastq3budget/properties (present)

users/admin/budgetapp/mypresent/bookmark/westq3budget/properties (present)

users/admin/budgetapp/present2/bookmark/mybudget/properties (present)

The code is as follows:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<!--import the following package in order to access the
    com.alphablox.blox.repository.BookmarkMatcherUsers class-->
<%@ page import="com.alphablox.blox.repository.*" %>
<html>
<head>
   <blox:header/>
</head>
<body>
<blox:bookmarks id="myBookmarksBlox" />
<%
   Bookmark bks[] = null;
   BookmarkMatcherUsers matcher = new BookmarkMatcherUsers();
   bks = null;
   matcher.setUser("admin");
   bks = myBookmarksBlox.listBookmarks(matcher);
%>
   <div>Got <%= bks.length %> Bookmark Object(s) for
        user <%= matcher.getUser() %></div>
   <div>The Bookmarks are:</div><br>
<%
   for (int i = 0; i < bks.length; i++) {
```

```
%><%= bks[i].getBinding() %> (<%= bks[i].getBloxType() %>)<br>
<%
      }
   %></div>
</body>
</html>
```

Example 4: Creating a bookmark using BookmarksBlox API

This example shows how to use a BookmarksBlox, Bookmark and BookmarkProperties classes to create a new bookmark. There are two ways to create a bookmark programmatically:

- Supply all the bookmark options to `BookmarksBlox.createBookmark(...)`
- Supply a Blox along with other information needed to `BookmarksBlox.createBookmark(...)`

This example demonstrates the later approach.

1. We specify the bookmark name, application name, user name, Blox name, visibility, and description associated with the bookmark.
2. Then we create a Bookmark object called "`bk`" using the `createBookmark()` method, and specify the Blox type to be `GRID_BLOX_TYPE`.
3. For the "`bk`" object, we create an instance of the BookmarkProperties object called "`gridBloxProp`" to store GridBlox specific properties and another called "`dataBloxProp`" to store DataBlox specific properties. For `gridBloxProp`, we set `cellBandingEnabled` to `true`; for `dataBloxProp`, we set the query to "`!`" and specify to reconnect to the data source.
4. Call the `saveAll()` method to save the bookmark we just created into the repository.

The generated output looks like the following:

(a GridBlox here)

```
We've got a Bookmark object from BookmarksBlox.createBookmark()!
Created a bookmark: q2fy02WestSales
At binding: users/jdoe/salesapp/mygrid/bookmark/q2fy02westsales/properties
```

Here is the code:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<!--import the following package in order to access the
    com.alphablox.blox.repository.BookmarkMatcherUsers class-->
<%@ page import="com.alphablox.blox.repository.*" %>
<blox:header />
<blox:bookmarks id="myBookmarksBlox" />

<blox:grid id="myGrid" width="500" height="320">
   <blox:data dataSourceName="qcc-essbase" query="!"/>
</blox:grid>
<%
// (1) Specify the bookmark properties
   String bookmarkName = "q2fy02WestSales";
   String applicationName = "SalesApp";
   String userName = "jdoe";
   String bloxName = "myGrid";
   String visibility = myBookmarksBlox.PRIVATE_VISIBILITY;
   String description = "Bookmark for Q2FY02 West Region Sales";
   Bookmark bk = null;

// (2) Create a Bookmark object called "bk"
   bk = myBookmarksBlox.createBookmark(bookmarkName,
      applicationName, userName, bloxName, visibility,
```

```
      myBookmarksBlox.GRID_BLOX_TYPE);

%>
<p>We've got a Bookmark object from BookmarksBlox.createBookmark()!<p/>

<%
// (3) Set the bookmark's description and its GridBlox and DataBlox
//     properties
   bk.setDescription(description);
   bk.setCustomProperty("Report", "West Region Sales Report");

   BookmarkProperties gridBloxProp =
      bk.createBookmarkProperties(myBookmarksBlox.GRID_BLOX_TYPE);
   gridBloxProp.setProperty("bandingEnabled", true);
   BookmarkProperties dataBloxProp =
      bk.createBookmarkProperties(myBookmarksBlox.DATA_BLOX_TYPE);
   dataBloxProp.setProperty("connectOnStartup", true);
   dataBloxProp.setProperty("query", "!");

// (4) Save the bookmarks to the repository. Must call save() or
//     saveAll() to save the bookmark to the repository.
   bk.saveAll();

%>
Created a bookmark: <%= bookmarkName %><br>
      At binding: <%= bk.getBinding() %>
<%
   bk = null;
%>
```

Example 5: Using server-side bookmarkLoad event filter

This example demonstrates how to use the server-side event filters to perform custom tasks (in this example, we pop up a MessageBox notifying the name of the loaded bookmark) when the `bookmarkLoad` event is triggered.

1. To use server-side event filters, first add the specific event filter object using the common Blox method `addEventFilter()`.

   ```
   <blox:present id="myPresent">
   ...
   <%
     myPresent.addEventFilter(new LoadFilter() );
   %>
   </blox:present>
   ```

2. Then write your own class that implements the corresponding event filter object (`BookmarkLoadFilter`) and the corresponding method (`bookmarkLoad(BookmarkLoadEvent)`) that will be called with the event is triggered. This requires adding the `com.alphablox.blox.filter.*` package import statement in your JSP.

   ```
   public class LoadFilter implements BookmarkLoadFilter
   {
      public void bookmarkLoad( BookmarkLoadEvent bre )
      {
         //actions to take when the event is triggered
      }
   }
   ```

Here is the code:

```
<%@ page import="com.alphablox.blox.filter.*" %>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.repository.Bookmark,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel" %>
<%@ taglib uri="bloxtld" prefix="blox"%>
```

```
<html>
<head>
  <title>Bookmarks Filter Events</title>
  <!-- Blox header tag -->
  <blox:header/>
</head>
<body>
<blox:present id="myPresent" >
   <blox:data dataSourceName="QCC-Essbase" query="!"/>
<%
   myPresent.addEventFilter(new LoadFilter(myPresent.getBloxModel()));
%>
</blox:present>
</body>
</html>

<%!
public class LoadFilter implements BookmarkLoadFilter {
   BloxModel model;
   public LoadFilter (BloxModel model) {
      this.model = model;
   }
   public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception {
      Bookmark bookmark = ble.getBookmark();
      String name = bookmark.getName();
      StringBuffer msg = new StringBuffer("A bookmark called " + name + " is
loaded.");

      MessageBox msgBox = new MessageBox(msg.toString(), "Bookmark Loaded",
MessageBox.MESSAGE_OK, null);
      model.getDispatcher().showDialog(msgBox);
      }
   }
%>
```

Example 6: Getting a bookmark's query when it is loaded

This example demonstrates how to get the textual query stored with a bookmark when a bookmarkLoad event is triggered.

1. Use server-side event filters BookmarkLoadFilter to trigger our custom action when a bookmark is loaded. See "Example 5: Using server-side bookmarkLoad event filter" on page 60 for an example of the server-side event filter. Note that the event filter should be added inside the PresentBlox tag so the filter is only added once rather than each time the page is reloaded:

   ```
   <blox:present id="myPresent" ...>

     <% myPresent.addBookmarkLoadFilter(new LoadFilter()); %>

   </blox:present>
   ```

2. Set the DataBlox's `textualQueryEnabled` property to `true` to apply the textual query when the bookmark is loaded:

   ```
   <blox:present id="myPresent" ...>

     <blox:data
       ...
       textualQueryEnabled="true" />

     <% myPresent.addBookmarkLoadFilter(new LoadFilter()); %>

   </blox:present>
   ```

3. When a bookmark is loaded, get the textual query from the bookmark's SerializedMDBQuery object (for multidimensional data sources) or the

SerializedTextualQuery object (for relational data sources). SerializedMDBQuery has a `generateQuery()` method and SerializedTextualQuery has a `getQuery()` method that return the textual query. Note that the `generateQuery()` method only works for IBM DB2 OLAP Server or Hyperion Essbase.

Here is the complete code:

```
<%@ page import="com.alphablox.blox.filter.*,
                 com.alphablox.blox.repository.BookmarkProperties,
                 com.alphablox.blox.repository.SerializedQuery,
                 com.alphablox.blox.repository.SerializedTextualQuery,
                 com.alphablox.blox.repository.SerializedMDBQuery,
                 com.alphablox.blox.repository.Bookmark,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel" %>

<%@ taglib uri="bloxtld" prefix="blox"%>
<html>
<head> <title>Bookmarks Filter Events</title>
<blox:header/>

</head>
<body>
<blox:present id="myPresent" width="800" height="600">
  <blox:data dataSourceName="QCC-Essbase"
    query="<ROW (\"All Locations\") Central East West <COLUMN (\"All Time
Periods\") 2001 !"
    useAliases="true"
    textualQueryEnabled="true" />

  <% myPresent.addEventFilter(new LoadFilter(myPresent.getBloxModel())); %>

</blox:present>

</body>
</html>


<%! public class LoadFilter implements BookmarkLoadFilter
{
  BloxModel model;
  public LoadFilter (BloxModel model) {
    this.model = model;
  }

  public void bookmarkLoad( BookmarkLoadEvent ble ) throws Exception
  {
    Bookmark bookmark = ble.getBookmark();
    SerializedQuery sq = bookmark.getSerializedQuery();
    SerializedTextualQuery stq = null;
    SerializedMDBQuery smq = null;
    String query = null;
    if( sq instanceof SerializedTextualQuery )
    {
       stq = (SerializedTextualQuery)sq;
       query = stq.getQuery();
    }
    else if( sq instanceof SerializedMDBQuery )
      {
        smq = (SerializedMDBQuery)sq;
        query  = smq.generateQuery();
      }
     StringBuffer msg = new StringBuffer("query=" + query);

     MessageBox msgBox = new MessageBox(msg.toString(), "Bookmark Event
Filter Message", MessageBox.MESSAGE_OK, null);
     model.getDispatcher().showDialog(msgBox);
```

```
    }
}
%>
```

BookmarksBlox Tag Attributes

This section describes the tag attributes supported by BookmarksBlox. Tasks related to managing bookmarks typically require the use of BookmarksBlox API. For details on BookmarksBlox methods, see com.alphablox.blox.BookmarkBlox in the Javadoc documentation.

id

This is a common Blox tag attribute and property. For a complete description, see “id” on page 36.

bloxName

This is a common Blox tag attribute and property. For a complete description, see “bloxName” on page 32.

Chapter 7. ChartBlox Tag Reference

This chapter contains reference material for ChartBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

ChartBlox Overview

ChartBlox displays data in a wide variety of pie, bar, and line formats. Users can change chart attributes, such as chart type and orientation, through the ChartBlox graphical user interface.

Graphical User Interface

The ChartBlox graphical user interface (GUI) consists of a chart display area and optional chart controls. Users can right-click on a member (can be on the legend, labels, or in the chart itself) to bring up the right-click menu, which gives them data navigation options such as drill up, drill down, pivot, and hide/show members. To change chart types, axes placement, or configure data, users can access the Chart Options dialog via the menu bar's Chart > Options... menu.

Available Chart Types

The following table lists the valid names of all available chart types when the ChartBlox is rendered in the DHTML client. When using one of these names as a value for the `chartType` property, omit any parenthetical comments.

Note: Note the following about chart types:

- The `chartType` property takes only the text string as a value. Be sure to spell it exactly as it appears in this table.
- The `get/setChartTypeAsInt()` methods take the integer shown to the left of each chart name.

Integer Value	Chart
200	Vertical Bar, Side-by-Side, 3D Effect (the default)
201	Vertical Line, Absolute, 3D Effect
202	Vertical Area, Absolute, 3D Effect
0	3D Bar
17	Vertical Bar, Side-by-Side (or simply "Bar")
18	Vertical Bar, Stacked
19	Vertical Bar, Side-by-Side, Dual Axis

20	Vertical Bar, Stacked, Dual Axis
21	Vertical Bar, Side-by-Side, Bipolar
22	Vertical Bar, Stacked, Bipolar
24	Horizontal Bar, Side-by-Side
25	Horizontal Bar, Stacked
26	Horizontal Bar, Side-by-Side, Dual Axis
27	Horizontal Bar, Stacked, Dual Axis
28	Horizontal Bar, Side-by-Side, Bipolar
29	Horizontal Bar, Stacked, Bipolar
31	Vertical Area, Absolute
32	Vertical Area, Stacked
33	Vertical Area, Absolute, Bipolar
34	Vertical Area, Stacked, Bipolar
35	Vertical Area, Percentage
41	Vertical Line, Absolute (or simply "Line")
42	Vertical Line, Stacked
43	Vertical Line, Absolute, Dual Axis
44	Vertical Line, Stacked, Dual Axis
45	Vertical Line, Absolute, Bipolar
46	Vertical Line, Stacked, Bipolar
47	Vertical Line, Percentage
55	Pie
61	Scatter
67	Radar, Line
68	Radar, Area
85	Histogram, Vertical
86	Histogram, Horizontal
89	Bubble Chart
502	Dial (See "Dial Charts" on page 67 for more information on how to use dial charts).
503	Contribution, Vertical
504	Contribution, Horizontal
510	Waterfall
511	Pareto

Chart types that have 3D effects include:
- 3D Bar (this is basically a Vertical Bar, Side-by-Side chart with the depth optimized).
- Vertical Bar, Side-by-Side, 3D Effect (the default)
- Vertical Line, Absolute, 3D Effect

- Vertical Area, Absolute, 3D Effect

Some chart types require you to specify more than one data value per each charted element. The following table lists these charts along with how many data values per element they require and the ordering requirements.

Integer	Type	Data Values and Order
61	Scatter	Two values per marker: X and Y, in that order.
89	Bubble Chart	Three values per marker: X, Y, and Z, in that order.

Dial Charts

Dial charts require specifications of several parameters before they can be drawn. These include the starting and ending numbers of the dial and the step size. Several nested tags are available for specification of a dial chart. For details, see "Dial Chart Tag Reference" on page 134.

Contribution Charts

Contribution charts are similar to ″stacked waterfall″ charts. They provide a specialized means to visualize two related variable series. Contribution charts require the specifications of two variable data series: the base and the contribution. On a vertical contribution chart, each bar in the chart is comprised of a base member (or column) at the bottom and a contribution member (or column) stacked on top. See the tag attribute "contribution" on page 85 for more information.

Chart Axes

Depending on the chart type, a chart can include one ordinal axis and up to three numeric axes (X1, Y1, and Y2). The ordinal axis contains groups or categories, and an O1 axis is included in all chart types except bubble and scatter charts. An X1 axis is only included in bubble and scatter charts. A Y1 axis is included in all chart types except pies. A Y2 axis is only included in dual-axes charts.

Specifying Style

You can set the style for the chart's title, axis title, footnote, and label by specifying the font and foreground color. For example, the following tag attribute sets the title style to use bold, 24-point Arial font with the color of #990099 and footnote style to use italic, 14-point Monospace font in red.

```
<blox:chart id="myChart"
   titleStyle="font=Arial:Bold:24, foreground=#990099"
   footnoteStyle="font=Monospace:Italic:14, foreground=red"
/>
```

You can also specify the style using the related nested tags as follows:

```
<blox:chart id="myChart">
   <blox:titleStyle
      font="Arial:Bold:24"
      foreground="#990099"
   />
   <blox:footnoteStyle
      font="Monospace:Italic:14"
      foreground="red"
   />
</blox:chart>
```

Setting the title, footnote, label, or axis title style overrides the defaults in the underlying theme.

Font

The font attribute takes the font name, style, and point size, separated by a colon:

font name: style: *point*

where:

- *font name*: These are defined according to the operating system. The following font names are generally accepted:
  - Arial
  - Courier
  - Helvetica
  - TimesRoman
  - SansSerif
  - Serif
  - Monospace

  Acceptable font values vary widely by browser and client machine. Therefore, generic names are provided (Monospace, SansSerif, and Serif). Each browser defines the actual font it will substitute for a generic name.

  If no font is specified, the default is SansSerif. If the server is running on a non-western language system, some characters may not display correctly if they cannot be found in the font's character set. To avoid this problem, always specify a font that will display correctly in your locale.
- *Style*: The font style can be one of the following values:
  - `plain`
  - `italic`
  - `bold`
  - `bolditalic`
- *Point*: An integer for point size (usually 8-36).

Tip: If no font is specified, the default is SansSerif. If the server is running on a non-Western language system, some characters may not display correctly if they cannot be found in the font's character set. To avoid this problem, always specify a font that will display correctly in your locale. Properties that involve font specification include axisTitleStyle, labelStyle, footnoteStyle, and titleStyle.

Tip: If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included, as shown in the following table:

Attribute Value	Result
`font=:Bold:12`	Uses the current font, but changes the style to bold, the size to 12 points.
`font=Helvetica`	Changes the font to Helvetica, but does not change the style or size.
`font=::14`	Uses the current font and style, but changes the size to 14 points.
`font=:Plain:`	Uses the current font and size, but changes the style to Plain.

Foreground

Color names are case-insensitive. The recognized color names are:

```
Black Blue Cyan DarkGray Gray Green          Magenta Orange Pink Red White Yellow
LightGray
```

A color can also be expressed as an RGB value that specifies the intensity of red, green, and blue, respectively, in a color. To specify a color by RGB value, convert each 3-number value to its hexadecimal or decimal equivalent. Then enter the resulting string, beginning a hexadecimal string with a number sign (#). For example, #00FF00 is the hexadecimal string for 100% green. The RGB, hexadecimal, and decimal values for the recognized color names are listed in the following table.

Tip: Check the web for palettes of browser-safe colors, such as:

- http://www.visibone.com/colorlab/

Color Name	RGB Value	Hex Value	Decimal Value
Black	000 000 000	000000	0
Blue	000 000 255	0000FF	255
Cyan	000 255 255	00FFFF	65535
DarkGray	064 064 064	404040	4210752
Gray	128 128 128	808080	8421504
Green	000 255 000	00FF00	62580
LightGray	192 192 192	C0C0C0	12632256
Magenta	255 000 255	FF00FF	16711935
Orange	255 200 000	FFC800	16762880
Pink	255 175 175	FFAFAF	16756655
Red	255 000 000	FF0000	16711680
White	255 255 255	FFFFFF	16777215
Yellow	255 255 000	FFFF00	16776960

ChartBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a ChartBlox. For a copy and paste version of the tag with all the attributes, see "ChartBlox JSP Custom Tag" on page 458.

Parameters

```
<blox:chart
    [attribute="value"] >
       [<blox:axisTitleStyle
              [attribute="value"] />]
       [<blox:dial
              [attribute="value"] />
       [<blox:footnoteStyle
              [attribute="value"] />]
       [<blox:labelStyle
              [attribute="value"] />]
       [<blox:seriesFill
```

```
            [attribute="value"] />]
      [<blox:titleStyle
            [attribute="value"] />]
</blox:chart>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
absoluteWarning
aggregateIdenticalInstances
applyPropertiesAfterBookmark
areaSeries
autoAxesPlacement
axisTitleStyle
backgroundFill
barSeries
bloxEnabled
bloxName
bookmarkFilter
chartAbsolute
chartCurrentDimensions
chartFill
chartType
columnLevel
columnSelections
comboLineDepth
contribution
contributionParameters
dataTextDisplay
dataValueLocation
depthRadius
dwellLabelsEnabled
enablePoppedOut
filter
footnote
footnoteStyle
formatProperties
gridLineColor
gridLinesVisible
groupSmallValues
height

Attribute
helpTargetFrame
histogramOptions
labelStyle
legend
legendPosition
lineSeries
lineWidth
logScaleBubbles
markerShape
markerSizeDefault
maxChartItems
maximumUndoSteps
menubarVisible
mustIncludeZero
noDataMessage
o1AxisTitle
pieFeelerTextDisplay
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
quadrantLineCountX
quadrantLineCountY
quadrantLineDisplay
removeAction
render
rightClickMenuEnabled
riserWidth
rowHeaderColumn
rowLevel
rowSelections
rowsOnXAxis
seriesColorList
showSeriesBorder
smallValuePercentage
title
titleStyle
toolbarVisible
totalsFilter
useSeriesShapes
visible

Attribute
width
x1AxisTitle
x1FormatMask
x1LogScale
x1ScaleMax
x1ScaleMaxAuto
x1ScaleMin
x1ScaleMinAuto
XAxis
XAxisTextRotation
y1Axis
y1AxisTitle
y1FormatMask
y1LogScale
y1ScaleMax
y1ScaleMaxAuto
y1ScaleMin
y1ScaleMinAuto
y2Axis
y2AxisTitle
y2FormatMask
y2LogScale
y2ScaleMax
y2ScaleMaxAuto
y2ScaleMin
y2ScaleMinAuto

<blox:axisTitleStyle> nested tag **See "axisTitleStyle" on page 78.**
Attribute
font
foreground

<blox:contribution> nested tag **See "contribution" on page 85.**
Attribute
baseSeries
contributionSeries
rootGroup
sortType

<blox:contribution> nested tag **See "contribution" on page 85.**
Attribute
thresholdType
thresholdValue

<blox:footnoteStyle> nested tag **See "footnoteStyle" on page 90.**
Attribute
font
foreground

<blox:labelStyle> nested tag **See "labelStyle" on page 94.**
Attribute
font
foreground

<blox:seriesFill> nested tag **See "seriesFill" on page 106.**
Attribute
index
value

<blox:titleStyle> nested tag **See "titleStyle" on page 108.**
Attribute
font
foreground

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

When there are no nested tags (such as the <blox:titleStyle> or <blox:footnoteStyle> tag), you can substitute the closing </blox:chart> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
width="650" />
```

When there are nested tags, the shorthand notation is not valid and a closing tag is required.

Examples

```
<blox:chart
     height="400"
     width="400"
     chartType="bar" />
```

ChartBlox Tag Attributes by Category

The following tables list the unique ChartBlox tag attributes. For lists of tag attributes common to several Blox, see "Common Blox Tag Attributes by Category" on page 29. The properties and methods supported by ChartBlox are organized in the cross reference as follows:

- "Chart Appearance Related Tag Attributes" on page 74
- "Chart Data Related Tag Attributes" on page 75
- "Chart Label Tag Attributes" on page 76
- "Chart Popped Out Tag Attributes" on page 76
- For dial charts, see "Dial Charts Overview" on page 129

Chart Appearance Related Tag Attributes

The following lists tag attributes related to the appearance of a chart.

Chart Appearance

- areaSeries
- autoAxesPlacement
- backgroundFill
- barSeries
- chartFill
- chartType
- comboLineDepth
- dataTextDisplay
- depthRadius
- footnoteStyle
- formatProperties
- gridLineColor
- gridLinesVisible
- histogramOptions
- labelStyle
- lineSeries
- lineWidth
- markerShape
- markerSizeDefault
- quadrantLineCountX
- quadrantLineCountY
- quadrantLineDisplay
- removeAction
- rightClickMenuEnabled
- riserWidth
- seriesColorList

- seriesFill
- showSeriesBorder
- titleStyle
- trendLines
- useSeriesShapes

Chart Data Related Tag Attributes

The following tag attributes relate to the data in a chart:

- absoluteWarning
- aggregateIdenticalInstances
- chartAbsolute
- chartCurrentDimensions
- columnLevel
- rowLevel
- columnSelections
- rowSelections
- dataValueLocation
- filter
- groupSmallValues
- legend
- logScaleBubbles
- maxChartItems
- mustIncludeZero
- rowsOnXAxis
- smallValuePercentage
- totalsFilter
- x1FormatMask
- x1LogScale
- x1ScaleMax
- x1ScaleMaxAuto
- x1ScaleMin
- x1ScaleMinAuto
- XAxis
- y1Axis
- y2Axis
- y1FormatMask
- y2FormatMask
- y1LogScale
- y2LogScale
- y1ScaleMax
- y1ScaleMaxAuto
- y2ScaleMax
- y2ScaleMaxAuto
- y1ScaleMin
- y1ScaleMinAuto

- y2ScaleMin
- y2ScaleMinAuto

Chart Label Tag Attributes

The following tag attributes relate to the labels that appear on a chart:

- axisTitleStyle
- dwellLabelsEnabled
- footnote
- footnoteStyle
- labelStyle
- legendPosition
- o1AxisTitle
- pieFeelerTextDisplay
- rowHeaderColumn
- title
- titleStyle
- XAxisTextRotation
- x1AxisTitle
- y1AxisTitle
- y2AxisTitle

Chart Popped Out Tag Attributes

The following are tag attributes related to displaying ChartBlox in a separate, popped out browser window.

- enablePoppedOut
- poppedOut
- poppedOutHeight
- poppedOutTitle
- poppedOutWidth

ChartBlox Tag Attributes

This section describes ChartBlox tag attributes. The attributes are listed alphabetically. For ChartBlox methods, see the com.alphablox.blox.ChartBlox class in the Javadoc documentation.

id

This is a common Blox property and tag attribute. For a detailed description, see "id" on page 36.

absoluteWarning

When the user sets chart values to display absolute values and at least one data value is negative, this warning is appended to the chart's footnote.

Data Sources

All

Syntax

```
absoluteWarning="warning"
```

where:

Argument	Default	Description
warning	"Warning: Values are absolute"	Warning message string.

aggregateIdenticalInstances

Specifies whether to group identical series and chart the aggregated data as one series, rather than showing each row in the grid as a separate series in the chart.

Data Sources

Relational data only

Syntax

```
aggregateIdenticalInstances="aggregate"
```

where:

Argument	Default	Description
aggregate	true	When this property is set to true, if there are instances where the series are identical, their aggregated data is charted. When it is set to false, each row in the grid is displayed individually as separate series.

applyPropertiesAfterBookmark

This is a common Blox property and tag attribute. For a detailed description, see "applyPropertiesAfterBookmark" on page 30.

areaSeries

Specifies which data series in a combination chart should be the area series.

Data Sources

All

Syntax

```
areaSeries="series"
```

where:

Argument	Default	Description
series	empty string	Comma-delimited string defining the displayed member names in the area series

When displaying an area, bar, or line chart, you can display the chart as a combination of these three chart types. It is possible to make one data series a line and another a bar and a third an area.

This property identifies the members represented on the area chart type as part of a combination chart. The displayed member names are defined as a

comma-delimited string. If there are multiple dimensions making up the legend item (as defined in Chart Axes Placement), you must use tabs ("\t") to separate the dimensions.

Examples

```
myPresent.getChartBlox().setAreaSeries("Qtr1\tAudio, Qtr2\tAudio,
Qtr3\tAudio");
```

See Also

"barSeries" on page 80, "lineSeries" on page 96

autoAxesPlacement

Determines how information should be placed on chart axes.

Data Sources

All

Syntax

```
autoAxesPlacement="auto"
```

where:

Argument	Default	Description
auto	true	Valid values are true or false.

Usage

The default indicates that ChartBlox should use the normal defaults for placing data on the x axis, legend, filter, y1 axis, and y2 axis. If you want to explicitly set any of these axes, legend or filter, this property should be set to false.

Examples

```
autoAxesPlacement="false"
```

See Also

"filter" on page 89, "legend" on page 95, "XAxis" on page 118, "y1Axis" on page 119, "y2Axis" on page 124

axisTitleStyle

Specifies the style (foreground colors and text format) for the chart axis title.

Data Sources

All

Syntax

```
axisTitleStyle="style"
```

or

```
<blox:axisTitleStyle
    font=""
    foreground="">
</blox:axisTitleStyle>
```

where:

Argument	Default	Description
`style`	empty string	String defining style attributes.

Usage
For details on how to specify the style string, see “Specifying Style” on page 67.

Examples
```
axisTitleStyle = "foreground=white, font=Courier:Bold:10"
```

See Also
“footnoteStyle” on page 90, “labelStyle” on page 94, “titleStyle” on page 108, “backgroundFill” on page 79

backgroundFill
Allows you to specify a solid color, color gradients, or images as the fill for the area outside of the chart frame.

Data Sources
All

Syntax
```
backgroundFill="fill"
```

where:

Argument	Default	Description
`fill`	null	String defining the color, gradient, or image for the chart background area.

Usage
The string `fill` can be either a solid color, a list of two colors for color gradients, or a URL to an image you want to display in the background. Specify the two colors using either standard Java color names or RGB values. If you use RGB values, enter them in the form `0xffffff`. If you want to use gradient colors, the string should be a comma-separated list of two colors. You can specify a gradient direction as the last item in the string by adding the appropriate gradient qualifier from the table below.

Gradient Direction	Qualifier
Right	1 (the default if a direction is not specified with a list of two colors)
Left	2
Down	3
Up	4
Down/Left	5
Up/Left	6
Down/Right	7

Up/Right 8

If you want to specify an image to use, it must be of one of the following formats:

- A relative URL from the application context to the image. For example, if your JSP resides in an application called "salesApp" and you want to use the image file `logo.gif` in the `salesApp/images/` directory for the background:

  ```
  backgroundFill = "images/logo.gif"
  ```

- An absolute URL that starts with "`http:`"

  ```
  backgroundFill = "http://serverName/path/to/image.gif"
  ```

  Note that the server where the referenced image file is located should not require authentication. If authentication is required, the image will not load and the default series color will be used. This is because the charting engine does not have a username and password to be authenticated.

- A URL that starts with "`file:`" using the file protocol:

  ```
  backgroundFill = "file:///C|/Alphablox5/webapps/salesApp/images/logo.gif"
  ```

  This is the file path to the image on the server where DB2 Alphablox is running.

The image will tile by default. If you want the image to stretch to fill the area, add:

```
, stretch"
```

to the end of the URL.

Examples

The following example fills the background with a solid color:

```
backgroundFill = "red"
```

The following example fills the background with a gradient from blue to green, with a direction that goes down to the right.

```
backgroundFill = "blue, green, 7"
```

The following example fills the background with a gradient from yellow to green. Since a direction is not specified, the default is from left to right.

```
backgroundFill = "yellow, green"
```

The following example stretches an image to fill the background:

```
backgroundFill = "images/logo.gif, stretch"
```

See Also

"chartFill" on page 82, "seriesFill" on page 106

barSeries

Specifies which data series in a combination chart should be the bar series.

Data Sources

All

Syntax

```
barSeries="series"
```

where:

Argument	Default	Description
series	empty string	Comma-delimited string defining the displayed member names in the bar series.

Usage

When displaying an area, bar, or line chart, you can display the chart as a combination of these three chart types. It is possible to make one data series a line and another a bar and a third an area.

This property identifies the members represented on the area chart type as part of a combination chart. The displayed member names are defined as a comma-delimited string. If there are multiple dimensions making up the legend item (as defined in Chart Axes Placement), you should use tabs ("\t") to separate the dimensions.

Examples

```
barSeries = "Qtr1\tVideo, Qtr2\t Video, Qtr3\tVideo"
```

See Also

"areaSeries" on page 77, "lineSeries" on page 96

bloxEnabled

This is a common Blox property and tag attribute. For a complete description, see "bloxEnabled" on page 32.

bloxName

This is a common Blox property and tag attribute. For a complete description, see "bloxName" on page 32.

bookmarkFilter

This is a common Blox property and tag attribute. For a complete description, see "bookmarkFilter" on page 31.

chartAbsolute

Specifies whether negative values should be treated as positive.

Data Sources

All

Syntax

```
chartAbsolute="chartAbsolute"
```

where:

Argument	Default	Description
chartAbsolute	false	A boolean value. Specify true for negative values to be treated as positive values, otherwise specify false.

Usage
In pie charts, for example, a negative value will not appear. By setting this property to `true`, the value appears as a positive value in the chart.

Tip: When one or more chart values is negative, ChartBlox displays a warning message. To modify the text of the message, use the absoluteWarning property.

Examples
```
chartAbsolute ="true"
```

See Also
"absoluteWarning" on page 76

chartCurrentDimensions
Specifies the current members to be used for the chart filters.

Data Sources
All

Syntax
```
chartCurrentDimensions="members"
```

where:

Argument	Default	Description
`members`	null	An array of strings which are the currently selected chart filter items. When setting the members, the members have to be in the same order the dimensions are in the chart's page filter. For example, if Products and Locations are on the chart's page filter, you can specify "Coke, East" to be the selected members.

chartFill
Allows you to specify a solid color or an image as the fill for the area inside the chart frame that is not the data representation.

Data Sources
All

Syntax
```
chartFill="fill"
```

where:

Argument	Default	Description
`fill`	null	String defining the color or image for the chart background area. The default is `#F0F0F0` (very light grey).

Usage
The string `fill` can be either a solid color or a URL to an image you want to display in the background. Specify the color using either standard Java color names or RGB values. If you use a RGB value, enter it in the form `0xffffff`.

If you want to specify an image to use, it must be of one of the following formats:

- A relative URL from the application context to the image. For example, if your JSP resides somewhere in an application context called "salesApp" and you want to use the image file `logo.gif` in the `salesApp/images/` directory, you should specify the following relative URL:

  ```
  chartFill = "images/logo.gif"
  ```

- An absolute URL that starts with "`http:`"

  ```
  chartFill ="http://serverName/path/to/image.gif"
  ```

  Note that the server where the referenced image file is located should not require authentication. If authentication is required, the image will not load and the default color will be used. This is because the charting engine does not have a username and password to be authenticated.

- A URL that starts with "`file:`" using the file protocol:

  ```
  chartFill ="file:///C|/DB2Alphablox/webapps/salesApp/images/logo.gif"
  ```

  This is the file path to the image on the server where DB2 Alphablox is running.

The image will tile by default. If you want the image to stretch to fill the area, add:

```
, stretch"
```

to the end of the URL.

Examples

```
chartFill = "red"
chartFill = "http://someServer/images/mypicture.gif"
chartFill = "file:///C|/Alphablox5/webapps/salesApp/images/logo.gif, stretch"
```

See Also

"footnoteStyle" on page 90, "labelStyle" on page 94, "titleStyle" on page 108, , "seriesFill" on page 106

chartType

Identifies the type of chart to display.

Data Sources

All

Syntax

```
chartType="type"
```

where:

Argument	Default	Description
`type`	`"Vertical Bar, Side-by-Side, 3D Effect"`	See "Available Chart Types" on page 65.

Usage

Frequently used values include "3D Bar", "Bar", "Pie", and "Line". The value must exactly match one of the entries in the table of "Available Chart Types" on page 65.

Note: The best way to view various types is to create a simple application page with a ChartBlox on it. Then invoke the application and use the **Chart Type** dialog box to preview chart types.

Examples

```
chartType="Vertical Bar, Stacked"
```

columnLevel

Specifies the data generation that the chart should use.

Data Sources

All

Syntax

```
columnLevel="levels"
```

where:

Argument	Default	Description
`levels`	none	A comma-separated list of integers specifying a set of dimension levels, where level 0 is the parent of every level. The first integer is the level for the first dimension in the column axis, the second integer is the level for the second dimension, and so on.

Usage

Setting for this property requires that the `totalsFilter` property be set to 2.

Examples

The following example sets the generation level to 2 for the first dimension placed on the column axis, and 4 for the second dimension.

```
columnLevel="2, 4"
```

See Also

"rowLevel" on page 103, "totalsFilter" on page 110

columnSelections

Specifies a subset of data to be charted.

Data Sources

All

Syntax

```
columnSelections="selections"
```

where:

Argument	Default	Description
`selections`	null	A String consisting of semicolon separated tuples, where the members of the tuples are separated by commas.

Usage

The value is a string consisting of a list of tuples separated by semicolons, with the members of each tuple separated by commas. Both `columnSelections` and `rowSelections` are set automatically when the user selects data in the grid and chooses to chart selected data, but they can be defined in the Blox so that the chart displays the specified data when loaded in the DHTML client. You must set both

the rowSelections and columnSelections properties in order for the chart to display data. If one is not set, the chart will be empty.

The default value of null indicates that all the data is charted.

Note: If your member names have commas or semicolons in them, you need to put double-quotes around each member name and escape the double-quotes, as follows:

```
columnSelections="\"East\", \"Qtr1\"; \"East\", \"Qtr2\""
```

Examples

```
columnSelections="East, Qtr1; East, Qtr2"
rowSelections="Actual, Audio; Actual, Visual"
```

See Also

"rowSelections" on page 104. For an additional example, see "Scriptlets Containing Blox APIs" on page 18.

comboLineDepth

Specifies the line depth in a combo chart.

Data Sources

All

Syntax

```
comboLineDepth="depth"
```

where:

Argument	Default	Description
depth	0	The depth of lines in a combo chart in pixels.

contribution

Specifies the parameters for a contribution chart. Similar to ″stacked waterfall″ charts, contribution charts provide a means to offer visualization of two related variable series. A contribution chart requires the specifications of two variable data series: the base and the contribution. On a vertical contribution chart, each bar in the chart is comprised of a base member (or column) at the bottom and a contribution member (or column) stacked on top.

Data Sources

All

Syntax

```
<blox:contribution
   baseSeries="baseSeries"
   contributionSeries="contributionSeries"
   rootGroup="rootGroup"
   sortType="sortType"
   thresholdType="thresholdType"
   thresholdValue="thresholdValue">
</blox:contribution>
```

where:

Argument	Default	Description
baseSeries	none	The scope for the base series. Specification of a base series is required. For example, for multidimensional data sources: `baseSeries="{Measures:sales}"` `baseSeries="{[My Cube].[Scenario]:[Actual]"` For relational data sources, use "{columns: *columnName* [,*columnName2*,...]}": `baseScope="{columns:sales}"`
contributionSeries	none	The scope for the contribution series. Specification of a contribution series is required. For example, for multidimensional data sources: `contributionSeries="{Measures:COGs}"` `contributionSeries="{[My Cube].[Year]:[Qtr1]}` For relational data sources, use "{columns: *columnName* [,*columnName2*,...]}". For example: `baseScope="{columns:increments}"`
rootGroup	none	The scope for the root group for comparison. The sum of the values for the base and the contribution member in the specified root group association is used in sorting regardless of sort type. That is, this attribute lets you arbitrarily compare the sum of base and contribution values in this root group with other values even when the `sortType` is set to `base` or `contribution`. This is an advanced feature, and is only needed when the underlying data requires programmatic intervention.
sortType	none	Type of sorting to perform. Valid values are `base`, `contribution`, `both`, or `none` (default).
thresholdType	none	Type of threshold. Valid values are `percent` , `count`, or `none` (default). When a threshold type is specified, the threshold value must also be specified. For example, to display only data for stores that contributed to 80% of the sales, `thresholdType` should be set to `percent`, and `thresholdValue` should be set to 80. To display data for the top 10 best selling products, `thresholdType` should be set to `count`, and `thresholdValue` should be set to 10.
thresholdValue	none	The threshold value. When a threshold value is specified, the threshold type must also be specified.

Examples

```
<blox:chart chartType="Contribution, Vertical">
   <blox:contribution
    thresholdType="count"
    thresholdValue="4"
    sortType="contribution"
    baseScope="{columns:base}"
    contributionScope="{columns:increment}">
   </blox:contribution>
</blox:chart>
```

contributionParameters

Specifies the parameters for a contribution chart.

Data Sources
All

Syntax
`contributionParameters="params"`

where:

Argument	Default	Description
`params`	none	A semicolon-separated string of attribute-value pairs. For example, `contributionParameters="sortType=contribution; thresholdType=count;thresholdValue=4.0; baseSeries={columns:base}; contributionSeries={columns:increment};"` For details on available attributes for a contribution chart, see "contribution" on page 85.

dataTextDisplay
Controls whether data values will be shown above each bar in a bar or waterfall chart.

Data Sources
All

Syntax
`dataTextDisplay="display"`

where:

Argument	Default	Description
`display`	`false`	A boolean argument. A value of `true` sets the display so the data values appear above the bars in a bar chart or waterfall chart. A value of `false` indicates that data values do not appear.

Examples
```
dataTextDisplay="true"
```

dataValueLocation
Specifies the dimension name and list of member names used in the chart.

Data Sources
All

Syntax
`dataValueLocation="data"`

where:

Argument	Default	Description
data	null	String of the form: "Dimension:Member1, Member2..." If there are multiple dimensions on an axis, use "\t" to separate the dimensions and members on the dimensions: "Dim1\tDim2...:Member1ofDim1\tMember1ofDim2, Member2ofDim1\tMember2ofDim2..."

Usage

When charting multidimensional data, this property defines which data to use on chart types that require more than one data value per element. When charting relational data, you must always use this property to define what data to chart. Use columns with numerical data only. If the columns contain other data, the chart will use null values and the chart will not be meaningful.

The syntax for data is the dimension name followed by a colon and a comma separated list of member names. When there is more than one dimension on an axis (for example, in a bubble chart), you can use "\t" to separate the dimensions and the members:

```
dataValueLocation="Scenario\tMeasures: Var% LY\tFS Sales,
   Act\tPromo %, Act\tFS Sales"
```

In the above example, two dimensions make up the axis: Scenario and Measures. The dimensions are separated with "\t" between them, and the members to be used are also specified with "\t" between them.

With relational data, the name of the column dimension is always "Columns". For examples:

```
dataValueLocation="Columns: Product1, Product2"
```

In order for certain charts to work correctly, you must define the data values in a specific order. The order depends on the type of chart you are using. For a listing of chart types and data value requirements, see "Available Chart Types" on page 65.

depthRadius

Sets the depth of the 3D effect on 2D charts.

Data Sources

All

Syntax

depthRadius="*radius*"

where:

Argument	Default	Description
radius	0	An integer between 0 to 100, indicating the degree of 3D effect.

Usage
The default value, 0, eliminates the 3D effect. The higher the value, the more pronounced the 3D effect.

Examples
```
depthRadius="45"
```

dwellLabelsEnabled
Specifies whether a dwell label (a text description of a data value) should appear when the user moves the mouse over a chart element.

Data Sources
All

Syntax
```
dwellLabelsEnabled="enabled"
```

where:

Argument	Default	Description
`enabled`	`true`	A boolean argument. A value of `true` indicates that the mouse-over labels appears on the chart, a value of `false` indicates that the labels do not appear.

Examples
```
isDwellLabelsEnabled();
setDwellLabelsEnabled(false);
```

enablePoppedOut
This is a property and tag attribute inherited from ContainerBlox. When the ChartBlox is nested within a PresentBlox:

- If the `poppedOut` property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the `poppedOut` property and its related properties have not been specified in the PresentBlox, the popped out settings in the nested ChartBlox are applied to the PresentBlox.

For a complete description, see “enablePoppedOut” on page 152.

filter
Specifies the dimensions that appear on the chart dimension filter.

Data Sources
All

Syntax
```
filter="filter"
```

where:

Argument	Default	Description
`filter`	empty string	Comma-delimited string defining filter dimensions.

Usage
ChartBlox determines the dimension placement if you use the default.
The `setFilter()` method automatically refreshes the chart.

```
filter="Product"
```

footnote
Specifies the text to appear in the chart footnote (at the bottom right of the chart).

Data Sources
All

Syntax
```
footnote="text"
```

where:

Argument	Default	Description
`text`	empty string	Any string. The text appears in the chart footnote.

Examples
```
footnote="Company Confidential"
```

See Also
"footnoteStyle" on page 90

footnoteStyle
Specifies the style (foreground colors and text format) for the footnote.

Data Sources
All

Syntax
```
footnoteStyle="style"
```

or

```
<blox:footnoteStyle
    font=""
    foreground="">
</blox:footnoteStyle>
```

where:

Argument	Default	Description
`style`	empty string	String defining style attributes.

Usage
For details on how to specify the style string, see “Specifying Style” on page 67.

Examples
```
footnoteStyle="foreground=white, font=Courier:Bold:10"
```

See Also
“axisTitleStyle” on page 78, “footnote” on page 90, “labelStyle” on page 94, “titleStyle” on page 108

formatProperties
Specifies chart format properties string to override the defaults. These format properties are used by the DHML client user interface to set colors, styles, and other attributes of the chart, such as data series colors or x-axis text rotation custom angle.

Data Sources
All

Syntax
`formatProperties="`*`formatProperties`*`"`

where:

Argument	Default	Description
`formatProperties`	empty string	Argument should be formatted as a comma-separated string of object property/value strings. Each object's property/value string should have each *property*:*value* pair separated by a semicolon, enclosed in curly braces. For example: *`ObjectName={property1:value1; property2:value2;},`* *`ObjectName2={property4:value4; property5:value5;}"`* For properties that involve multiple values, separate the values with comma: *`ObjectName1={property1:value1; property2:value2a,value2b;},`* *`ObjectName2={property3:value3a, value3b; property4:value4a,value4b;}"`* Alternatively, you can also use scope strings as keys. For example: *`{Dim0:Mem0}{Dim1:Mem1}= {property0:value0,value1;property1:value2;},`* *`{Dim2:Mem2,Mem3}={property2:value3;}`* See “Scoping” on page 173 in the DataBlox's calculatedMembers section for the syntax on scope strings.

Usage
This property is currently only used to set individual data series colors, x-axis text rotation custom angle, and waterfall color array. Everything else should be set through normal named chart properties.

Examples

```
formatProperties="colorSeries_default_0 = {foreground:yellow;},
colorSeries_default_1 = {foreground:red;},
colorSeries_default_4 = {foreground:#FF9900;},
chart={XAxisTextRotation:45;}"
```

gridLineColor

Sets the color of the grid line.

Data Sources

All

Syntax

```
gridLineColor="color"
```

where:

Argument	Default	Description
color	black	The name or hexadecimal value of a color.

Usage

The default grid line color when the application is rendered in DHTML client is #D0D0D0 (light grey). For more information on Java colors, see http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Color.html.

Examples

```
gridLineColor="red"
gridLineColor="#00ffff"
```

gridLinesVisible

Specifies whether lines appear underlaid on a two-dimensional chart.

Data Sources

All

Syntax

```
gridLinesVisible="enabled"
```

where:

Argument	Default	Description
visible	true	Specify true to display grid lines; false to hide them.

Usage

Some charts, such as a 3D bar chart, do not display grid lines, even if gridLinesVisible is set to true.

Examples

```
gridLinesVisible="false"
```

groupSmallValues

Groups smaller values into an “Other” item on a pie chart. This property affects only pie charts.

Data Sources
All

Syntax
`groupSmallValues="groupSmall"`

where:

Argument	Default	Description
`groupSmall`	`true`	A boolean argument. A value of `true` indicates that values that are too small to chart are grouped into an "other" category on the chart, a value of `false` indicates that they will be charted.

Usage
Pie charts with many smaller values can be difficult to read, and grouping items together can improve chart readability.

The minimum percentage used for this grouping is set by the `smallValuePercentage` property.

Examples
```
groupSmallValues="false"
```

See Also
"smallValuePercentage" on page 108

height
This is a common Blox property and tag attribute. For a complete description, see "height" on page 35.

helpTargetFrame
This is a common Blox property and tag attribute. For a complete description, see "helpTargetFrame" on page 35.

histogramOptions
Sets the options for histogram charts.

Data Sources
All

Syntax
`histogramOptions="options"`

where:

Argument	Default	Description
options	binMode=basic; useSize=false; binCount=6; addCumm=false; sort=false	A semicolon-delimited string of option and value pairs. Valid options are: • addCumm: true or false; true to add a cumulative percentage line to the chart (as in Pareto chart). The default is false. • binCount: the number of bins to include in the chart • binList: a comma-delimited list of numerics. For custom binning (binMode=custom) used to explicitly set bin ranges. Each number in the list is the inclusive, upper value for a bin. • binMode: basic (default) or custom. In basic mode, the bins are set either through binCount or binSize. The value set in binCount determines the bin ranges. The value set in binSize determines the number of bins in the chart. • binSize: the size of bin used to sort values. It should be a positive numeric value. • maxBin: the biggest value to store in the last (highest) bin. Used in conjunction with either binCount or binSize to determine bin ranges or number of bins. If this is not set, the largest value in the data is used. • minBin: the lowest value to store in the last (lowest) bin. Used in conjunction with either binCount or binSize. If this is not set, the lowest value in the data is used. • useSize: true or false. For basic binning (binMode=basic), when useSize is true, bins are created using binSize. Otherwise, bins are created based on binCount. • sort: true or false. A value of true results in a descending sort. When this option is combined with addCumm (sort=true;addCumm=true), it creates a Pareto chart.

Examples

The following example creates a sorted histogram chart with 10 bins and the chart includes a cumulative percentage line.

```
<blox:chart histogramOptions="addCumm=true;sort=true;binCount=10" .../>
```

labelStyle

Specifies the style (foreground colors and font) for the chart labels.

Data Sources

All

Syntax

```
labelStyle="style"
```

or

```
<blox:labelStyle
    font=""
    foreground="">
</blox:labelStyle>
```

where:

Argument	Default	Description
style	empty string	String defining style attributes.

Usage
For details on how to specify the style string, see “Specifying Style” on page 67.

Examples
```
labelStyle="foreground=white, font=Courier:Bold:10"
```

See Also
“axisTitleStyle” on page 78, “footnoteStyle” on page 90,, “titleStyle” on page 108

legend
Specifies the dimensions that appear on the legend.

Data Sources
All

Syntax
```
legend="legend"
```

where:

Argument	Default	Description
legend	empty string	A comma-delimited string of dimensions.

Usage
ChartBlox determines the dimension placement if you use the default. The `setLegend()` method automatically refreshes the chart.

Examples
```
legend="Measures, Market"
```

See Also
“legendPosition” on page 95

legendPosition
Specifies if and where chart legends are to appear.

Data Sources
All

Syntax
```
legendPosition="position"
```

where:

Argument	Default	Description
position	bottom	A String with one of the following values:none, bottom, right. When the application's rendering mode is DHTML, the default is bottom. If the application's rendering mode is set to Java, the default is right. For dial charts, when you specify the dials using the nested <blox:dial> tag, legendPosition is automatically set to none as the legend will not be meaningful to users. See "Creating a Dial Chart" on page 130 for more information.

Usage
Valid values are:
- none – do not display legends.
- bottom – display legend beneath the chart.
- right – display legend to the right of the chart.

The default is bottom if the application's default render mode is set to DHTML.

Examples
```
legendPosition="none"
```

See Also
"legend" on page 95

lineSeries
Specifies which data series in a combination chart should be the line series.

Data Sources
All

Syntax
```
lineSeries="series"
```

where:

Argument	Default	Description
series	empty string	A comma-delimited string defining the displayed member names in the line series.

Usage
When displaying an area, bar, or line chart, you can display the chart as a combination of these three chart types. It is possible to make one data series a line and another a bar and a third an area.

This property identifies the members represented on the area chart type as part of a combination chart. The displayed member names are defined as a comma-delimited string. If there are multiple dimensions making up the axis (as defined in Chart Axes Placement), you must use tabs (\t) to separate the dimensions.

Examples

```
lineSeries="Qtr1\tAll Products, Qtr2\tAll Products, Qtr3\tAll Products"
```

See Also

"areaSeries" on page 77, "barSeries" on page 80

lineWidth

Controls the width of lines drawn on line charts.

Data Sources

All

Syntax

```
lineWidth="width"
```

where:

Argument	Default	Description
`width`	3	A positive integer defining the line width in pixels.

Examples

```
lineWidth="7"
```

logScaleBubbles

Specifies whether to use a logarithmic scale to set bubble sizes in bubble charts.

Data Sources

All

Syntax

```
logScaleBubbles="useLogScale"
```

where:

Argument	Default	Description
`useLogScale`	`false`	`true` to set bubble sizes using a logarithmic scale.

markerShape

Sets the shape of the markers.

Data Sources

All

Syntax

```
markerShape="shape"
```

where:

Argument	Default	Description
shape		A comma-separated list of numbers. Valid values are: • 0 = null • 1 = Square • 2 = Circle • 3 = Diamond • 4 = Plus • 5 = Triangle/Down • 6 = Triangle/Up

Usage
The shapes will repeat. Setting the property to ″1,3,4″ will result in a square for the marker for the first series, the second a diamond, the third a plus, and then the fourth will be a square and so on.

Examples
```
markerShape="1, 3,4"
```

markerSizeDefault
Sets the size of the marker which appears on line charts and bubble charts.

Data Sources
All

Syntax
```
markerSizeDefault="size"
```

where:

Argument	Default	Description
size	30	Valid values are 0 to 100. A value of 30 is about 10 pixels. A value of 100 is about 30 pixels.

Examples
```
markerSizeDefault="10"
```

maxChartItems
Sets the maximum number of items allowed in the chart result set. If the result set exceeds this number, chart generation stops and you get an error message.

Data Sources
All

Syntax
```
maxChartItems="items"
```

where:

Argument	Default	Description
items	256	A positive integer indicating the maximum items that can be charted.

Usage
Some charts might become difficult to read after a certain number of items are charted. This property is useful for limiting the number of items to be charted. The actual number of items you can chart and still be able to read the chart effectively will get larger as the size of your chart gets larger.

Examples
```
maxChartItems="10"
```

maximumUndoSteps
This is a common Blox property and tag attribute. For a complete description, see "maximumUndoSteps" on page 36.

menubarVisible
This is a common Blox property and tag attribute. For a complete description, see "menubarVisible" on page 37.

mustIncludeZero
Specifies whether to include zero on the chart axes.

Data Sources
All

Syntax
```
mustIncludeZero="includeZero"
```

where:

Argument	Default	Description
includeZero	true	A boolean argument. A value of true indicates that zero is charted, a value of false indicates that zero is not charted.

Usage
If set to true, there will always be a zero on the chart axes; the chart will begin counting at zero, regardless of the actual starting point on the measurement. If set to false, the measurement begins at a point close to the smallest value on the chart. In order to use log scale ([axis]LogScale) or [axis]ScaleMin, mustIncludeZero must be set to false.

Examples
```
mustIncludeZero="false"
```

noDataMessage
This is a common Blox property and tag attribute. For a complete description, see "noDataMessage" on page 37.

o1AxisTitle

Explicitly defines the title for the O1 axis.

Data Sources

All

Syntax

```
o1AxisTitle="title"
```

where:

Argument	Default	Description
`title`	null	Any string, indicating the text for the axis title.

Usage

O1 axis is the first ordinal axis in a chart that contains groups or categories. See "Chart Axes" on page 67 for details on chart axes. When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and an empty string will display no title. A returned value of `null` for the getter method signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples

```
o1AxisTitle="This is the O1 Axis"
```

See Also

"x1AxisTitle" on page 114, "y1AxisTitle" on page 120, "y2AxisTitle" on page 125

pieFeelerTextDisplay

For pie charts, this property specifies whether and how pie slices should be labeled.

Data Sources

All

Syntax

```
pieFeelerTextDisplay="type"
```

where:

Argument	Default	Description
`type`	1	An integer between 0 to 3, inclusive.

Usage

Valid values and their meanings are:

`0` = Do not label pie slices.

`1` = Show each text label at the end of a "feeler line" (a line that extends from the pie slice to the text).

2 = Show labels only, with no feeler lines. Labels are positioned just outside the slice.
3 = Place labels directly on the pie slices.

Examples

```
pieFeelerTextDisplay="3"
```

poppedOut

This is a property inherited from ContainerBlox. When the ChartBlox is nested within a PresentBlox:

- If the `poppedOut` property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the `poppedOut` property and its related properties have not been specified in the PresentBlox, the popped out settings in the ChartBlox are applied to the PresentBlox.

For a complete description, see "poppedOut" on page 153.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 154.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 154.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 155.

quadrantLineCountX

Sets the number of vertical lines that will appear on a bubble chart. It is ignored for all other chart types.

Data Sources

All

Syntax

```
quadrantLineCountX="count"
```

where:

Argument	Default	Description
`count`	1	The value cannot be less than 1.

Usage

To remove the quadrant lines altogether, use the quadrantLineDisplay property.

Examples

```
quadrantLineCountX="2"
```

See Also
"quadrantLineCountY" on page 102, "quadrantLineDisplay" on page 102

quadrantLineCountY
Sets the number of horizontal lines that will appear on a bubble chart. It is ignored for all other chart types.

Data Sources
All

Syntax
```
quadrantLineCountY="count"
```

where:

Argument	Default	Description
count	1	The value cannot be less than 1.

Usage
To remove the quadrant lines altogether, use the quadrantLineDisplay property.

Examples
```
quadrantLineCountY="2"
```

See Also
"quadrantLineCountX" on page 101, "quadrantLineDisplay" on page 102

quadrantLineDisplay
Sets whether or not to display quadrant lines on a bubble chart.

Data Sources
All

Syntax
```
quadrantLineDisplay="display"
```

where:

Argument	Default	Description
display	true	To display no quadrant lines, set this property to false.

Examples
```
quadrantLineDisplay="false"
```

See Also
"quadrantLineCountX" on page 101, "quadrantLineCountY" on page 102

removeAction
This is a common Blox property. For a complete description, see "removeAction" on page 38.

render

This is a common Blox property. For a complete description, see “render” on page 38.

rightClickMenuEnabled

This is a common Blox property. For a complete description, see “rightClickMenuEnabled” on page 39.

riserWidth

Sets the width of the risers in bar charts. This value is a percentage of the available space.

Data Sources

All

Syntax

```
riserWidth="width"
```

where:

Argument	Default	Description
width	75	The percentage of the available space. A value of 100 would leave no space between the different groups.

Examples

```
riserWidth="85"
```

rowHeaderColumn

Defines which column contains the names of the row labels and which column the chart uses to create the axis labels.

Data Sources

Relational

Syntax

```
rowHeaderColumn="name"
```

where:

Argument	Default	Description
name	null	A String containing a column name.

Usage

Used only for relational data.

Examples

```
rowHeaderColumn="Column header name"
```

rowLevel

Returns the data generation that the chart should use.

Data Sources
All

Syntax
`rowLevel="levels"`

where:

Argument	Default	Description
`levels`	none	A comma-separated list of integers specifying a set of dimension levels, where level 0 is the parent of every level.

Usage
This method requires that the `totalsFilter` property be set to 2.

Examples
```
rowLevel="3, 1"
```

See Also
"columnLevel" on page 84, "totalsFilter" on page 110

rowsOnXAxis
When enabled, swaps the chart axes such that data appear on opposite axes.

Data Sources
Relational

Syntax
`rowsOnXAxis="rowsOnXAxis"`

where:

Argument	Default	Description
`rowsOnXAxis`	`false`	A boolean argument. A value of `true` indicates that the row axis is charted on the X axis, a value of `false` indicates that the row axis is charted on the Y axis.

Examples
```
rowsOnXAxis="true"
```

rowSelections
Specifies a subset of data to be charted.

Data Sources
All

Syntax
`rowSelections="selections"`

where:

Argument	Default	Description
selections	none	A String consisting of semicolon separated tuples, where the members of the tuples are separated by commas.

Usage

The value is a string consisting of a list of tuples separated by semicolons, with the members of each tuple separated by commas. These properties are set automatically when the user selects data in the grid and chooses to chart selected data, but they can be defined in the Blox so that the chart displays the specified data when loaded in the DHTML mode. You must set both the rowSelections and columnSelections properties in order for the chart to display data. If one is not set, the chart will be empty.

The default value of null indicates that all the data is charted.

Note: If your member names have commas or semicolons in them, you need to put double-quotes around each member name and escape the double-quotes, as follows:

```
rowSelections="\"Actual\", \"Audio\"; \"Actual\", \"Visual\""
```

Examples

```
columnSelections="East, Qtr1; East, Qtr2"
rowSelections="Actual, Audio; Actual, Visual"
```

See Also

"columnSelections" on page 84

seriesColorList

Sets the list of colors that will be used when charting the current series.

Data Sources

All

Syntax

```
seriesColorList="list"
```

where:

Argument	Default	Description
list	null	A comma-delimited list of colors.

Usage

Sets the list of colors that will be used when charting the current series. The colors are specified in a comma separated string, and can be standard Java color names or hexadecimal values. The default color when the application is rendered in DHTML is "#9691C7", "#00B09B", "#68AEE0", "#008B87", "#99CCCC", "#005699", "#C2C4C6", "#998300", "#CCAE99", "#A76100", "#E0CB68", "#B03400".

Be sure to list enough colors for the data you are charting. If you do not define enough colors, the colors you specify will be repeated in sequence for the remaining series.

Examples

```
seriesColorList = "red, green, blue, #FFCCFF"
```

See Also

"seriesFill" on page 106

seriesFill

Specifies color gradients or images as the fill for the bars, lines, or areas that represent the data within the chart. This API takes an `index` argument which indicates the series of data in the chart to which you are applying the fill.

Data Sources

All

Syntax

```
<blox:seriesFill
   index=""
   value="" >
</blox:seriesFill>
```

where:

Argument	Default	Description
`index`	null	An integer representing the number of any data series present.
`value`	null	String defining the color gradient, or image for the specified data series.

Usage

The `value` tag attribute/method argument is a string that can be either a list of two colors for color gradients or a URL to an image you want to display in an area. Specify the colors using either standard Java color names or RGB values. If you use RGB values, enter them in the form `0xffffff` or `#ffffff`. If you want to use gradient colors, the string should be a comma-separated list of two colors.

If you want the bars to contain gradients, you can specify a gradient direction as the last item in the string by adding the appropriate gradient qualifier from the table below. If you want the bars to contain solid colors, do not set the `seriesFill` property; instead, set the seriesColorList property.

Gradient Direction	Qualifier
Right	1 (the default if a direction is not specified)
Left	2
Down	3
Up	4
Down/Left	5
Up/Left	6
Down/Right	7
Up/Right	8

If you skip indices when specifying the seriesFill, the skipped series will use the same seriesFill as the last specified seriesFill. In the following example, series 2 and 3 will use the same gradient as series 1:

```
<blox:seriesFill index="1" value="green, yellow"/>
<blox:seriesFill index="4" value="blue, green"/>
```

If you want to specify an image to use, it must be of one of the following formats:

- A relative URL from the application context to the image. For example, if your JSP resides in an application called "salesApp" and it uses the image file logo.gif in the salesApp/images/ directory:

  ```
  <blox:seriesFill index="1" value="images/logo.gif" />
  ```

- An absolute URL that starts with "http:"

  ```
  <blox:seriesFill index="2" value="http://serverName/path/to/image.gif" />
  ```

  Note that the server where the referenced image file is located should not require authentication. If authentication is required, the image will not load and the default series color will be used. This is because the charting engine does not have a username and password to be authenticated.

- A URL that starts with "file:" using the file protocol:

  ```
  <blox:seriesFill index="2"
  value="file:///C|/DB2Alphablox/webapps/salesApp/images/logo.gif" />
  ```

  This is the file path to the image on the server where DB2 Alphablox is running.

The image will always tile. If you use a transparent GIF image, it will overlay the series color (see "seriesColorList" on page 105).

Examples

```
<blox:chart ...>
   <blox:seriesFill index="1" value="green, yellow, 2"/>
</blox:chart>
```

The example above sets the first data series to a gradient of green to yellow, with a direction that goes to the left.

See Also

"footnoteStyle" on page 90, "labelStyle" on page 94, "titleStyle" on page 108, "chartFill" on page 82, "seriesColorList" on page 105

showSeriesBorder

Specifies whether or not a border is shown around the bars on a chart and the legend squares.

Data Sources

All

Syntax

showSeriesBorder="*show*"

where:

Argument	Default	Description
show	false	true — to show border around bars and legend squares; false— not to show border. The default is false.

Usage
Applies to bar charts only. The default is `false` when the application's rendering mode is set to DHTML.

smallValuePercentage
Sets the minimum percentage to group smaller values into an "Other" item on a pie chart. This property affects only pie charts.

Data Sources
All

Syntax
`smallValuePercentage="`*`percentage`*`"`

where:

Argument	Default	Description
`percentage`	`5.0`	An argument of type double. Valid values are between 0.01 and 10.0, inclusive.

Usage
Pie charts with many smaller values can be difficult to read, and grouping items together can improve chart readability.

Examples
```
smallValuePercentage"7.2"
```

See Also
"groupSmallValues" on page 92

title
Specifies the text to appear as a title above the chart.

Data Sources
Multidimensional

Syntax
`title="`*`text`*`"`

where:

Argument	Default	Description
`style`	empty string	String defining style attributes.

Examples
```
title="My Title"
```

See Also
"titleStyle" on page 108

titleStyle
Specifies the style (foreground colors and text format) for the chart's title.

Data Sources
All

Syntax
```
titleStyle="style"
```

or

```
<blox:titleStyle
    font=""
    foreground="">
</blox:titleStyle>
```

where:

Argument	Default	Description
style	empty string	String defining style attributes.

Usage
For details on how to specify the style string, see “Specifying Style” on page 67.

Examples
```
titleStyle="foreground=white, font=Courier:Bold:10"
```

See Also
“axisTitleStyle” on page 78, “footnoteStyle” on page 90, “labelStyle” on page 94, “title” on page 108

toolbarVisible
Specifies if the toolbar is visible.

Data Sources
All

Syntax
JSP Tag Attribute

```
toolbarVisible="visible"
```

where:

Argument	Default	Description
visible	true	true— the toolbar is visible; false— the toolbar is not visible. When the application’s rendering mode is set to DHTML, toolbar is visible by default.

Usage
By default, the toolbar is visible in a standalone ChartBlox. If a nested <blox:toolbar> tag is added, its setting overwrites the value of this attribute. For example, the following code will result in a visible toolbar.

```
<blox:chart id="myChart" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:chart>
```

Tip: `toolbarVisible` is only a tag attribute, not a property.

totalsFilter

Specifies if and how totals appear in a chart.

Data Sources

Multidimensional

Syntax

`totalsFilter="type"`

where:

Argument	Default	Description
type	1	An integer between 0 and 2, inclusive.

Usage

Valid values and their meanings are:

`0` = No filter; all totals appear.
`1` = Show the last (lowest-drilled-to) generation of each dimension on the chart.
`2` = Show a user-specified generation.

Tip: A value of "2" causes a set of radio buttons to appear at the bottom of the chart, with which the user selects the generation to appear on the chart. To limit the scope of the generation available, see the properties "rowLevel" on page 103 and "columnLevel" on page 84.

Examples

```
totalsFilter="0"
```

See Also

"rowLevel" on page 103, "columnLevel" on page 84

trendLines

Creates trendlines in the chart.

Data Sources

All

Syntax

`trendLines="trendLines"`

where:

Argument	Default	Description
`trendLines`	null	A comma-separated string of trendlines: `trendLines="[trendLine1],[trendLine2],...,[trendLineN]"` Each trendline is a semicolon separated string of *parameter=value* pairs: `param1=value1;param2=value2;...;paramN=valueN` Valid parameters are: • `name`: Required. A string for the name of the trendline. • `member`: Required. One or more `member` parameter can be added. The member to use to plot the trendline, in the form of *uniqueDimensionName*: *uniqueMemberName* For example: `member=All Locations:East` • `type`: Required. Valid types are: – `exponential` – `linear` – `logarithmic` – `moving average(N)`: where N is at least 2 – `polynomial(N)`: where N is greater or equal to 2 and less or equal to 100. – `power` • `drilldownscope`: Optional. Valid values are `descendents` and `none`. • `replace`: Optional. Set to `true` for the trendline to be drawn replacing the line or bar with which it is associated; `false`, not to replace. • `forecastforward`: Optional. A number of periods or units to forecast. Not supported for moving average trendline type. • `color`: Optional. Specify a Java color or a hexadecimal value (for example, #CCCCFF). • `style`: Optional. Set to `solid` or `dashed` for the line style.

Usage

Trendlines can be added to line, bar, area or scatter charts. The trendlines added can be modified by end users via the **Chart** > **Trendlines...** option from the menu bar.

The following table describes the types of trendlines supported:

Type	Description	Equation
Linear	The least squares fit for a line represented.	$y = c_0 + c_1 x$ where C1 is the slope and C0 is the intercept.
Logarithmic	The least squares fit through the data points.	$y = c_0 + c_1 \ln x$ where C0 and C1 are constants, and ln is the natural logarithm function.

Type	Description	Equation
Polynomial	The least squares fit through the data points.	$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_n x^n$ where C0 and C1...Cn are constants, where N is greater than or equal to 2, and N is less than or equal to 100.
Power	The least squares fit through the data points.	$y = cx^b$ where c and b are constants
Exponential	The least squares fit through the data points.	$y = ce^{bx}$ where c and b are constants, and e is the base of the natural logarithm.
Moving Average	The average over the a specified time period. The number of periods in a moving average trendline equals the total number of points in the series less the number you specify for the period.	$F_{(t+1)} = \frac{1}{N} \sum_{j=0}^{N-1} A_{t-j+1}$ where N is the number of prior periods to include in the moving average; Aj is the actual value at time j; Fj is the forecasted value at time j.

You can also implement your own trendline algorithm.

Custom trendline algorithms

All trending algorithms used by DB2 Alphablox are described in the `trendingtypes.xml` file. This file contains the name and full class name for each trending algorithm This file contains the display name and the full class name for each trending algorithm. An entry is as follows:

```
<trend-type id="polynomial">
<display-name lang="en">Polynomial</display-name>
<class>com.alphablox.blox.uimodel.core.chart.trending.PolynomialFitAlgorithm</class>
</trend-type>
```

In order for the class to be loaded, the default class loader must be able to find the class. In addition, your custom trending algorithm class must extend com.alphablox.blox.uimodel.core.chart.trending.AbstractTrendingAlgorithm. See the Javadoc documentation of this class for further details.

Note: Custom trending algorithms will not be reflected in the trendline dialog.

A simple custom trending algorithm example

The following example implements all the methods required by AbstractTrendingAlgorithm. The custom function is defined in the `f(number x)` function. It returns 0 if the computation results in a null.

```
public class myTrendingAlgorithm extends AbstractTrendingAlgorithm {
   public void initialize(Number[] x, Number[] y, Chart chart) {
   }

   public Double f(Number x) {
```

```
      // create your algorithm
      return new Double(x == null ? 0 : (20 + (x.doubleValue() * 20)));
   }

   public boolean isForecastEnabled() {
      return true;
   }

   public void setUserParameter(Object parameter) {
   }

   public boolean isScatterSupported() {
      return true;
   }

   public void parseSavedParameters(String parameters) {
      super.parseSavedParameters(parameters);
   }

   public String getParameterString() {
      return super.getParameterString();
   }

   public boolean isNullAllowed() {
      return true;
   }
}
```

Installation of custom trending algorithms

After the custom trending algorithm is created:

1. Compile the class.
2. Create a JAR file containing your custom class.
3. Add the JAR file to your JVM's class path. For WebSphere, the JAR file should be placed under `<websphere_app_server_dir>/lib/ext`. For other Web application servers, see the topic on "Setting Class Path" in the *Administrator's Guide*'s "Extending DB2 Alphablox" section.
4. Edit the `trendingTypes.xml` file in `<alphablox_dir>/repository/servers` and add your trending algorithm.
5. Restart the server for the changes to take effect.

Examples

The following example creates two trendlines, one polynomial with an order of 3 and the other, linear.

```
trendLines="name=poly(3);member=All Locations:All Locations;
replace=true; type=polynomial(3),
name=line;member=All Locations:Central;type=linear"
```

This creates a polynomial trendline plotted for each member in All Locations, whereas the Central location has a linear trendline. The original data series lines/bars are replaced (`replace=true`).

useSeriesShapes

Sets whether the legend for a line chart displays the shapes used for the data points in the chart.

Data Sources

All

Syntax

```
useSeriesShapes="display"
```

where:

Argument	Default	Description
display	true	A boolean argument. A value of true specifies that the legend displays shapes for the different data points, a value of false specifies that the shapes are not displayed. The default is true when the application's rendering mode is set to DHTML.

Examples

```
useSeriesShapes="true"
```

visible

This is a common Blox property. For a complete description, see "visible" on page 40.

width

This is a common Blox property. For a complete description, see "width" on page 40.

x1AxisTitle

Explicitly defines the title for the X1 axis. This property only applies to bar charts, line charts, and area charts.

Data Sources

All

Syntax

```
x1AxisTitle="title"
```

where:

Argument	Default	Description
title	null	Any string, indicating the text for the axis title.

Usage

When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and an empty string will display no title. A returned value of "null" for the getter methods signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples

```
x1AxisTitle="This is the X1 Axis"
```

See Also

"o1AxisTitle" on page 100, "pieFeelerTextDisplay" on page 100, "XAxis" on page 118, "XAxisTextRotation" on page 119, "y1AxisTitle" on page 120, "y2AxisTitle" on page 125

x1FormatMask

Specifies the value of the format mask for the X1 axis of a scatter or bubble chart.

Data Sources

All

Syntax

```
x1FormatMask="formatMask"
```

where:

Argument	Default	Description
formatMask	null	See "Numeric Formatting" on page 221 and "formatMask" on page 246.

Usage

The format mask allows you to specify customized formatting for the chart axis values. This format is also used in the dwell labels that appear when the mouse pauses over a chart data item. For example, if the X1 axis measures a percentage, you can specify `##0.00%` for the format mask. The format masks are set with the same way as the format masks on the grid. For information about how to set the value for these properties, see "Numeric Formatting" on page 221. The keywords `K` and `M` (for thousands and millions) are supported. Division (/) and multiplication (*) are also supported.

Examples

```
x1FormatMask="##0.00%"
x1FormatMask="$#,###/1000"
xy1FormatMask="#,###K"
```

To set the Y1 axis format for dollar values rounded to the nearest 100 million:

```
y1FormatMask="$###,###,###.##"
```

See Also

"y1FormatMask" on page 121, "y2FormatMask" on page 125, "Numeric Formatting" on page 221

x1LogScale

Sets whether or not to use a logarithmic scale for the X1 axis.

Data Sources

All

Syntax

```
x1LogScale="width"
```

where:

Argument	Default	Description
logScale	false	Specifies whether to use a logarithmic scale for the X1 axis.

Usage

When x1LogScale is set to true, since the chart engine does not automatically calculate the maximum and minimum values for the scale, the following properties have to be specified as well:

- x1ScaleMaxAuto has to be set to false.
- A value for x1ScaleMax has to be specified.
- x1ScaleMinAuto has to be set to false.
- A value for x1ScaleMin has to be specified.
- mustIncludeZero has to be set to false.

Since a log scale cannot start with 0 and the values will never be negative, mustIncludeZero must be set to false (the default is true).

Examples

```
x1LogScale="true"
```

See Also

"x1ScaleMaxAuto" on page 117, "x1ScaleMax" on page 116, "x1ScaleMinAuto" on page 118, "x1ScaleMin" on page 117, "mustIncludeZero" on page 99, "y1LogScale" on page 121, "y2LogScale" on page 126

x1ScaleMax

Sets the maximum value of the X1 axis.

Data Sources

All

Syntax

```
x1ScaleMax="scale"
```

where:

Argument	Default	Description
scale	null	The maximum value of the X1 axis.

Usage

This property is ignored if x1ScaleMaxAuto is set to true. x1ScaleMax should always be set to a value larger than x1ScaleMin, or the chart may not behave properly.

Examples

```
x1ScaleMax="500000"
```

See Also

"x1ScaleMaxAuto" on page 117, "x1ScaleMin" on page 117

x1ScaleMaxAuto

Sets whether or not to automatically set the maximum value of the X1 axis. When this is `false`, the value of the x1ScaleMax property will be used to set the maximum value of the X1 axis.

Data Sources
All

Syntax
`x1ScaleMaxAuto="`*`auto`*`"`

where:

Argument	Default	Description
`auto`	`true`	Valid values are `true` or `false`. When this is set to `false`, the maximum value of the X1 axis will be determined by the value of the x1ScaleMax property. By default, this is set to `true` and the value of the x1ScaleMax property is ignored.

Usage
When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to `false` and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin.* Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples
```
x1ScaleMaxAuto="false"
```

See Also
"x1LogScale" on page 115, "x1ScaleMax" on page 116

x1ScaleMin

Sets the minimum value of the X1 axis.

Data Sources
All

Syntax
`x1ScaleMin="`*`scale`*`"`

where:

Argument	Default	Description
`scale`	null	The minimum value of the X1 axis.

Usage
This property is ignored if x1ScaleMinAuto is set to `true`. If `mustIncludeZero` is set to `true`, it has priority if `x1ScaleMin` is greater than zero. `x1ScaleMax` should always be set to a value larger than `x1ScaleMin`, or the chart may not behave properly.

Examples

```
x1ScaleMin="10000"
```

See Also

"x1ScaleMinAuto" on page 118, "x1ScaleMax" on page 116, "mustIncludeZero" on page 99

x1ScaleMinAuto

Sets whether or not to automatically set the minimum value of the X1 axis. When this is `false`, the value of the x1ScaleMin property will be used to set the minimum value of the X1 axis.

Data Sources

All

Syntax

```
x1ScaleMinAuto="auto"
```

where:

Argument	Default	Description
`auto`	`true`	Valid values are `true` or `false`. When this is set to `false`, the minimum value of the X1 axis will be determined by the value of the x1ScaleMin property. By default, this is set to `true` and the value of the x1ScaleMin property is ignored.

Usage

When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to `false` and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin*. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples

```
x1ScaleMinAuto="false"
```

See Also

"x1LogScale" on page 115, "x1ScaleMin" on page 117

XAxis

Specifies the dimensions on the X axis. This property only applies to bar charts, line charts, and area charts.

Data Sources

All

Syntax

```
XAxis="xAxis"
```

where:

Argument	Default	Description
xAxis	empty string	A comma-delimited string of dimension names.

Usage
When using the default, ChartBlox determines the dimension placement. The setXAxis() method automatically refreshes the chart.

See "Chart Axes" on page 67 for details on chart axes

Examples
```
xAxis="All Products"
```

See Also
"x1AxisTitle" on page 114, "x1FormatMask" on page 115, "XAxisTextRotation" on page 119, "y1Axis" on page 119, "y2Axis" on page 124

XAxisTextRotation
Specifies the X axis label rotation. This property only applies to bar charts, line charts, and area charts.

Data Sources
All

Syntax
```
XAxisTextRotation="type"
```

where:

Argument	Default	Description
type	0	An integer between 0 and 2, inclusive.

Usage
The accepted values are:
- 0 = Normal
- 1 = Rotated 90 degrees
- 2 = Staggered

Examples
```
xAxisTextRotation="2"
```

See Also
"x1AxisTitle" on page 114, "XAxis" on page 118

y1Axis
Specifies the member names on the Y1 axis.

Data Sources
All

Syntax

```
y1Axis="y1Axis"
```

where:

Argument	Default	Description
y1Axis	empty string	Comma-delimited string of member display names. The members should be from the same dimension.

Usage

See "Chart Axes" on page 67 for details on chart axes.

Under the default, ChartBlox determines the dimension placement. The `setY1Axis()` method automatically refreshes the chart.

Examples

```
y1Axis="Market"
```

See Also

"y1AxisTitle" on page 120, "y1FormatMask" on page 121, "y2Axis" on page 124

y1AxisTitle

Explicitly defines the title for the Y1 axis.

Data Sources

All

Syntax

```
y1AxisTitle="title"
```

where:

Argument	Default	Description
title	null	Any string, indicating the text for the axis title.

Usage

See "Chart Axes" on page 67 for details on chart axes.

When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and an empty string will display no title. A returned value of "null" for the getter methods signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples

```
y1AxisTitle="This is the Y1 Axis"
```

See Also

"o1AxisTitle" on page 100, "pieFeelerTextDisplay" on page 100, "x1AxisTitle" on page 114, "y1Axis" on page 119, "y1FormatMask" on page 121, "y2AxisTitle" on page 125

y1FormatMask

Specifies the value of the format mask for the Y1 axis on a chart.

Data Sources

All

Syntax

```
y1FormatMask="formatMask"
```

where:

Argument	Default	Description
formatMask	null	See "Numeric Formatting" on page 221 and "formatMask" on page 246.

Usage

The format mask allows you to specify customized formatting for the chart axis values. This format is also used in the dwell labels that appear when the mouse pauses over a chart data item. For example, if the Y1 axis measures a percentage, you can specify `##0.00%` for the format mask. The format masks are set with the same way as the format masks on the grid. For information about how to set the value for these properties, see "Numeric Formatting" on page 221. The keywords `K` and `M` (for thousands and millions) are supported. Division (/) and multiplication (*) are also supported.

Examples

```
y1FormatMask="##0.00%"
y1FormatMask="$#,###/1000"
y1FormatMask="#,###K"
```

To set the Y1 axis format for dollar values rounded to the nearest 100 million:

```
y1FormatMask="$###,###,###.##"
```

See Also

"y2FormatMask" on page 125, "x1FormatMask" on page 115, "Numeric Formatting" on page 221

y1LogScale

Sets whether or not to use a logarithmic scale for the Y1 axis.

Data Sources

All

Syntax

```
y1LogScale="width"
```

where:

Argument	Default	Description
logScale	false	Specifies whether to use a logarithmic scale for the Y1 axis.

Usage

When y1LogScale is set to true, since the chart engine does not automatically calculate the maximum and minimum values for the scale, the following properties have to be specified as well:

- y1ScaleMaxAuto has to be set to false.
- A value for y1ScaleMax has to be specified.
- y1ScaleMinAuto has to be set to false.
- A value for y1ScaleMin has to be specified.
- mustIncludeZero has to be set to false.

Since a log scale cannot start with 0 and the values will never be negative, mustIncludeZero must be set to false (the default is true).

Examples

```
y1LogScale="true"
```

See Also

"y1ScaleMaxAuto" on page 123, "y1ScaleMax" on page 122, "y1ScaleMinAuto" on page 124, "y1ScaleMin" on page 123, "mustIncludeZero" on page 99, "y2LogScale" on page 126, "x1LogScale" on page 115

y1ScaleMax

Sets the maximum value of the Y1 axis.

Data Sources

All

Syntax

```
y1ScaleMax="scale"
```

where:

Argument	Default	Description
scale	null	The maximum value of the Y1 axis.

Usage

This property is ignored if y1ScaleMaxAuto is set to true. y1ScaleMax should always be set to a value larger than y1ScaleMin, or the chart may not behave properly.

```
y1ScaleMax="500000"
```

See Also

"y1LogScale" on page 121, "y1ScaleMaxAuto" on page 123, "y1ScaleMin" on page 123.

y1ScaleMaxAuto

Sets whether or not to automatically set the maximum value of the Y1 axis. When this is `false`, the value of the y1ScaleMax property will be used to set the maximum value of the Y1 axis.

Data Sources
All

Syntax
`y1ScaleMaxAuto="auto"`

where:

Argument	Default	Description
`auto`	`true`	Valid values are `true` or `false`. When this is set to `false`, the maximum value of the Y1 axis will be determined by the value of the y1ScaleMax property. By default, this is set to `true` and the value of the y1ScaleMax property is ignored.

Usage
When using log scale on an axis, all the corresponding `[axis]ScaleMaxAuto` and `[axis]ScaleMinAuto` have to be set to `false` and a value has to be specified for `[axis]ScaleMax` and `[axis]ScaleMin`. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples
`y1ScaleMaxAuto="false"`

See Also
"y1LogScale" on page 121, "y1ScaleMax" on page 122, "y1ScaleMin" on page 123, "y1ScaleMinAuto" on page 124

y1ScaleMin

Sets the minimum value of the Y1 axis.

Data Sources
All

Syntax
`y1ScaleMin="scale"`

where:

Argument	Default	Description
`scale`	null	The minimum value of the Y1 axis.

Usage
This property is ignored if y1ScaleMinAuto is set to `true`. `y1ScaleMax` should always be set to a value larger than `y1ScaleMin`, or the chart may not behave properly. `mustIncludeZeo` should be set to `false` since it has priority if `y1ScaleMin` is greater than zero

Examples
```
y1ScaleMin="10000"
```

See Also
"y1ScaleMinAuto" on page 124, "y1ScaleMax" on page 122

y1ScaleMinAuto

Sets whether or not to automatically set the minimum value of the Y1 axis. When this is `false`, the value of the y1ScaleMin property will be used to set the minimum value of the Y1 axis.

Data Sources
All

Syntax
```
y1ScaleMinAuto="auto"
```

where:

Argument	Default	Description
`auto`	`true`	Valid values are `true` or `false`. When this is set to `false`, the minimum value of the Y1 axis will be determined by the value of the y1ScaleMin property. By default, this is set to `true` and the value of the y1ScaleMin property is ignored.

Usage
When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to `false` and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin*. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples
```
y1ScaleMinAuto="false"
```

See Also
"y1LogScale" on page 121, "y1ScaleMin" on page 123

y2Axis

Specifies the member display names on the Y2 axis.

Data Sources
All

Syntax
```
y2Axis="y2Axis"
```

where:

Argument	Default	Description
`y2Axis`	empty string	Comma-delimited string of member display names. The members should be from the same dimension.

Usage
Under the default, ChartBlox determines the dimension placement. The `setY2Axis()` method automatically refreshes the chart. See “Chart Axes” on page 67 for details on chart axes.

Examples
```
y2Axis="Market"
```

See Also
“y1Axis” on page 119, “y2AxisTitle” on page 125, “y2FormatMask” on page 125

y2AxisTitle

Explicitly defines the title for the Y2 axis. See “Chart Axes” on page 67 for details on chart axes.

Data Sources
All

Syntax
```
y2AxisTitle="title"
```

where:

Argument	Default	Description
`title`	null	Any string, indicating the text for the axis title.

Usage
When charting relational data, the chart will not automatically display any axis titles. You must define all titles that you want displayed on the chart.

You can also specify titles with multidimensional data sources, but it is not required. The default value in this case, null, will automatically set axes titles, and an empty string will display no title. A returned value of “null” for the getter methods signifies that the chart automatically determined the axis titles from a multidimensional data source.

Examples
```
y2AxisTitle="This is the Y2 Axis"
```

See Also
“o1AxisTitle” on page 100, “pieFeelerTextDisplay” on page 100, “x1AxisTitle” on page 114, “y1AxisTitle” on page 120, “y2Axis” on page 124, “y2FormatMask” on page 125

y2FormatMask

Specifies the value of the format mask for the Y2 axis on a chart.

Data Sources
All

Syntax

```
y2FormatMask="formatMask"
```

where:

Argument	Default	Description
formatMask	null	See "Numeric Formatting" on page 221 and "formatMask" on page 246.

Usage

The format mask allows you to specify customized formatting for the chart axis values. This format is also used in the dwell labels that appear when the mouse pauses over a chart data item. For example, if the Y2 axis measures a percentage, you can specify `##0.00%` for the format mask. The format masks are set with the same way as the format masks on the grid. For information about how to set the value for these properties, see "Numeric Formatting" on page 221. The keywords `K` and `M` (for thousands and millions) are supported. Division (/) and multiplication (*) are also supported.

Examples

```
y2FormatMask="##0.00%"
y2FormatMask="$#,###/1000"
y2FormatMask="#,###K;
```

To set the Y2 axis format for dollar values rounded to the nearest 100 million:

```
y2FormatMask="$###,###,###.##"
```

See Also

"y1FormatMask" on page 121, "x1FormatMask" on page 115,"Numeric Formatting" on page 221

y2LogScale

Sets whether or not to use a logarithmic scale for the Y2 axis.

Data Sources

All

Syntax

```
y2LogScale="width"
```

where:

Argument	Default	Description
logScale	false	Specifies whether to use a logarithmic scale for the Y2 axis.

Usage

When `y2LogScale` is set to `true`, since the chart engine does not automatically calculate the maximum and minimum values for the scale, the following properties have to be specified as well:

- y2ScaleMaxAuto has to be set to `false`.
- A value for y2ScaleMax has to be specified.
- y2ScaleMinAuto has to be set to `false`.

- A value for y2ScaleMin has to be specified.
- mustIncludeZero has to be set to `false`.

Since a log scale cannot start with 0 and the values will never be negative, mustIncludeZero must be set to `false` (the default is `true`).

Examples

```
y2LogScale="true"
```

See Also

"y2ScaleMaxAuto" on page 127, "y2ScaleMax" on page 127, "y2ScaleMinAuto" on page 128, "y2ScaleMin" on page 128, "mustIncludeZero" on page 99, "x1LogScale" on page 115, "y1LogScale" on page 121

y2ScaleMax

Sets the maximum value of the Y2 axis.

Data Sources

All

Syntax

```
y2ScaleMax="scale"
```

where:

Argument	Default	Description
`scale`	null	The maximum value of the Y2 axis.

Usage

This property is ignored if y2ScaleMaxAuto is set to `true`. `y2ScaleMax` should always be set to a value larger than `y2ScaleMin`, or the chart may not behave properly.

Examples

```
y2ScaleMax="500000"
```

See Also

"y2ScaleMaxAuto" on page 127, "y2ScaleMin" on page 128

y2ScaleMaxAuto

Sets whether or not to automatically set the maximum value of the Y2 axis. When this is `false`, the value of the y2ScaleMax property will be used to set the maximum value of the Y2 axis.

Data Sources

All

Syntax

```
y2ScaleMaxAuto="auto"
```

where:

Argument	Default	Description
auto	true	Valid values are true or false. When this is set to false, the maximum value of the Y2 axis will be determined by the value of the y2ScaleMax property. By default, this is set to true and the value of the y2ScaleMax property is ignored.

Usage
When using log scale on an axis, all the corresponding [axis]ScaleMaxAuto and [axis]ScaleMinAuto have to be set to false and a value has to be specified for [axis]ScaleMax and [axis]ScaleMin. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to false.

Examples
```
y2ScaleMaxAuto="false"
```

See Also
"y2LogScale" on page 126, "y2ScaleMax" on page 127

y2ScaleMin
Sets the minimum value of the Y2 axis.

Data Sources
All

Syntax
```
y2ScaleMin="scale"
```

where:

Argument	Default	Description
scale	null	The minimum value of the Y2 axis.

Usage
This property is ignored if y2ScaleMin is set to true. y2ScaleMax should always be set to a value larger than y2ScaleMin or the chart may not behave properly. mustIncludeZeo should be set to false it has priority if y2ScaleMin is greater than zero.

Examples
```
y2ScaleMin="10000"
```

See Also
"y2ScaleMinAuto" on page 128, "y2ScaleMax" on page 127

y2ScaleMinAuto

Sets whether or not to automatically set the minimum value of the Y2 axis. When this is false, the value of the y2ScaleMin property will be used to set the minimum value of the Y2 axis.

Data Sources
All

Syntax
```
y2ScaleMinAuto="auto"
```

where:

Argument	Default	Description
auto	true	Valid values are true or false. When this is set to false, the minimum value of the Y2 axis will be determined by the value of the y2ScaleMin property. By default, this is set to true and the value of the y2ScaleMin property is ignored.

Usage
When using log scale on an axis, all the corresponding *[axis]ScaleMaxAuto* and *[axis]ScaleMinAuto* have to be set to `false` and a value has to be specified for *[axis]ScaleMax* and *[axis]ScaleMin*. Since a log scale cannot include 0 or negative numbers, mustIncludeZero has to be set to `false`.

Examples
```
y2ScaleMinAuto="false"
```

See Also
"y2LogScale" on page 126, "y2ScaleMin" on page 128

Dial Charts Overview

Custom JSP tags are available for defining a dial chart. A dial chart is a circular chart with one or more dials. They are often used in executive dashboards, flash reports, or Key Performance Indicator (KPI) types of scenarios. Each dial has a scale, a needle, and one or more dial sectors. A dial sector is used to highlight a specified region on a dial chart with a particular color. For instance, you might have a dial plotting inventory with a minimum dial value of 100 and a maximum dial value of 500. There could be a red dial sector for the region between 100 and 200 indicating that if the needle is in this region, there is a danger of a supply inventory shortage.

A dial chart can contain more than one dials; each dial can have its own needle with multiple sectors. Typically, you assign different colors to the regions. Each dial has a start angle, an end angle, a needle, and a radius. A radius of 100% means the greatest possible length allowed given the size of the chart area. The combination of start/end angles and radius makes it possible to put multiple dials on a dial chart. The Blox Sampler contains several dial chart examples that demonstrate the different properties of a dial.

By default, dial charts are plotted based on the first available data value unless you specify otherwise. The API for creating dial charts is available in the com.alphablox.blox.uimodel.core.chart.dial package. Custom JSP tags are available for specifying most of the common properties and are described in "Dial Chart Tag Reference" on page 134.

Creating a Dial Chart

To create a dial chart:

1. Set the ChartBlox `chartType` property to `dial`.
2. Add a nested `<blox:dial>` tag for each dial.
3. For each dial, specify its needle, sector, and scale. The hierarchy of the tags is as follows:

```
<blox:chart chartType="dial">
   <blox:dial ...>
      <blox:needle ...>
      <blox:sector ...>
      <blox:scale ...>
   </blox:dial>
</blox:chart>
```

These tags allow you to specify the attributes for each of the components in a dial chart. For detailed listing on the tags, see "Dial Chart Tag Reference" on page 134. For discussions on each of the dial chart component, see "Dial Chart Components" on page 130.

Note: When you add a nested `dial` tag inside `<blox:chart>`, the chart type will be forced to `dial` even if the `chartType` property is not set or set to something else.

Tip: When you add a dial chart using the tags, the `legendPosition` property of the ChartBlox is automatically set to `none`. Changing the legend position through the ChartBlox dialog in the user interface will have no effect since the legend will not show. You should not reset `legendPosition` to other positions, because the legend will not be meaningful to users.

Dial Chart Components

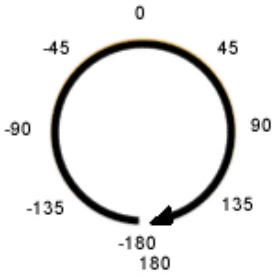
The key components in a dial chart include: Dial, Scale, Sector, and Needle.

Dial

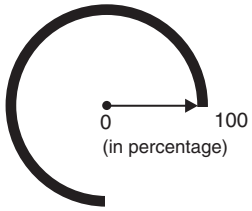
Each dial chart can contain one or more dials. A `<blox:dial>` tag is provided for you to specify the following key properties for each dial in your dial chart:

- `startAngle`: The position at which the dial region will begin.
- `stopAngle`: The position at which the dial region will stop.
- `radius`: The radius of the dial as a percentage of the available space.
- `color`: The fill color for this dial.
- `ticPosition`: The position for the tic marks on the dial. The position can be `inside`, `outside`, or `none`.
- `borderType`: The type of border for the dial. It can be `solid` or `none`.
- `borderColor`: The color for the dial's border.

The following diagram shows the how the values for `startAngle` and `stopAngle` are determined:



The following diagram shows how the `radius` is determined:



Scale

For each dial, you need to specify its scale. The scale for a dial consists of the minimum value, the maximum value, and the step size between tic marks on the dial. Any needles plotted on this dial are plotted against this scale. The dial's minimum and maximum values are automatically determined based on the first data value unless you specify otherwise. A nested `<blox:scale>` tag is provided that allows you to specify the following key attributes:

- `maximum`: The maximum value on the scale.
- `minimum`: The minimum value on the scale.
- `stepSize`: The step size of the tic markers.
- `scope`: The data value used to determine the values on this dial's scale.

The minimum and maximum values can be specified using either percentages or actual values. Using percentages is recommended as this allows the scale to be set dynamically based on the data value. If you specify actual values (without the "%" sign), the data value used for the needle may exceed the maximum value on the scale, resulting in problems plotting the needle. Only specify actual values in a "static" chart where data drilling action is disabled and the value for the needle will not change.

Sector

Using dial sectors, you can divide a dial into different sectors with different colors. This is useful for signalling threshold values. A dial sector is defined with:

- A start value and a stop value.
- An inner radius and an outer radius, which are expressed as percentages of the dial's radius.

A nested `<blox:sector>` tag is provided that allows you to specify the following key attributes:

- `startValue`: The beginning value for this sector. This can be either a percentage or actual data value, depending on how the minimum and maximum values are specified in the dial's scale.

- `stopValue`: The end value for this sector. This can be either a percentage or actual data value, depending on how the minimum and maximum values are specified in the dial's scale.
- `color`: The color for this sector.
- `innerRadius`: The inner radius for this sector. The default is 0, the center of the circle.
- `outerRadius`: The outer radius for this sector. The default is 100, the full length available.
- `scope`: The cell whose value should be used to determine the values in the sector.
- `tooltip`: The text to display when the mouse hovers over the sector.

The code snippet that generates the above output is available in "Example 1: Specifying Sectors" on page 132.

Needle

A needle is used in a dial chart to indicate what the actual, current data value is against some threshold numbers. Each dial can have one needle. A nested `<blox:needle>` tag is provided that allows you to specify the following key properties:

- `needleWidth`: The width of the needle in pixels.
- `endType`: The type of needle's end. The needle end can be a sharp arrow, a block arrow, a circle, or no needle end.
- `endWidth`: The width of the needle need.
- `color`: The color for this needle.
- `tooltip`: The tooltip to display when users mouse over the needle end.
- `scope` or `value:`The cell whose value, or the actual value, the needle should be pointing to.

Dial Chart Examples

The section includes complete examples that you can run to see the output. Examples include:

- "Example 1: Specifying Sectors" on page 132
- "Example 2: Specifying Needles and Scope" on page 133

Example 1: Specifying Sectors

The following example demonstrates how sectors are created inside a dial. For a live example, see the Blox Sampler.

1. The chart type is first set to "dial." A PresentBlox is used here to include the GridBlox to show you how the dial's scale is determined.
2. There are two dials in this chart. The first one has an angle of -150 to 150, and the second one is a complete circle with an angle of -180 to 180.
3. The scale for this dial goes from 0 to 150% of the first data value returned from the query, with a tic markers for each increment of 2500.
4. Four sectors are created inside this dial, each with a different color (red, yellow, green, and dark green).
5. The second dial in this chart. This one is simply to add a small circle in the center as the base for the needle for better appearance of the chart. Its `needleWidth` is set to `0` and `needleType` to `none`.

The complete code that generates the above output is as follows:

```
      <%@ taglib uri="bloxtld" prefix="blox"%>
      <html>
      <head>
         <blox:header/>
      </head>
      <body>

      <blox:present id="dialExample1" height="90%" width="90%">
         <blox:chart chartType="dial" y1FormatMask="$#,###"
            titleStyle="font=Arial:bold:10"
            title="Milk Chocolate Truffles Sales for Jan 01">
            <blox:dial startAngle="-150" stopAngle="150" color="#CCCC99">
(1)            <blox:scale minimum="0" maximum="150%" stepSize="2500" />

(2)            <blox:sector startValue="0" stopValue="50%"
(3)               color="red" innerRadius="30" outerRadius="80"
(4)               tooltip="Below expectation" />
               <blox:sector startValue="50%" stopValue="80%"
                  color="yellow" outerRadius="80"
                  tooltip="Marginal performance"/>
               <blox:sector startValue="80%" stopValue="120%"
                  color="green" innerRadius="80"
                  tooltip="Satisfactory performance"/>
               <blox:sector startValue="120%" stopValue="150%"
                   color="#009966" innerRadius="80" />
            </blox:dial>

            <%--creating a blue circle in the center --%>
            <blox:dial startAngle="-180" stopAngle="180" color="black"
               radius="10" ticPosition="none" borderType="none">
               <blox:needle needleWidth="0" endType="none" />
(5)         </blox:dial>
         </blox:chart>
         <blox:data dataSourceName="qcc-essbase" useAliases="true"
            query="<ROW (\"All Products\") <CHILD \"Truffles\"
               <COLUMN (\"All Time Periods\") <CHILD \"Qtr 1 01\" !" />
      </blox:present>
      </body>
      </html>
```

Example 2: Specifying Needles and Scope

The following example generates a dial chart with four dials and four different needle types.

1. The chart type is first set to "dial."
2. The first dial in the chart is simply to create a color border. The start and end angles are set to -135 to 135 with no tic markers or needles.
3. The second dial is the actual dial to convey meaningful data. Its radius is set to 90 so the first dial becomes the border.
4. The scale for the second dial is based on the value of `{scenario:forecast}`, with the minimum value being 0 and the maximum value being 120% of Forecast ($16,828,805 is 120% of the forecasted $14,024,008).
5. The needle for the second dial is based on the value of `{scenario:actual}`.
6. The yellow sector ends and the green sector starts at 100% of the forecasted value. This allows the users to see whether the actual value has achieved the forecasted goal.
7. The third dial in the chart is simply to add a small circle in the center as the base for the needle for better appearance of the chart. Its `needleWidth` is set to `0` and `needleType` to `none`.

The complete code that generates the above output is as follows:

```
       <%@ taglib uri="bloxtld" prefix="blox"%>
       <html>
       <head>
          <blox:header/>
       </head>
       <body>
       <blox:present id="dialExample1" height="90%" width="90%">
          <blox:chart chartType="dial" y1FormatMask="$#,###">
             <blox:dial startAngle="-135" stopAngle="135" color="#CCCC99"
                ticPosition="none" showLabels="false">
                <blox:needle needleWidth="0" endType="none" />
(1)          </blox:dial>
(2)
             <blox:dial startAngle="-135" stopAngle="135" radius="90">
                <blox:scale minimum="0" maximum="120%"
                   scope="{scenario:forecast}" />
(3)             <blox:needle color="blue" needleWidth="3"
(4)                endType="sharpArrow" scope="{scenario:actual}" />
                <blox:sector startValue="0" stopValue="75%"
(5)                color="red" />
                <blox:sector startValue="75%" stopValue="100%"
                   color="yellow"/>
                <blox:sector startValue="100%" stopValue="120%"
(6)                color="green" />
             </blox:dial>

             <%--creating a blue circle in the center --%>
             <blox:dial startAngle="-180" stopAngle="180" color="black"
                radius="10" ticPosition="none" borderType="none">
                <blox:needle needleWidth="0" endType="none" />
(7)          </blox:dial>
          </blox:chart>
          <blox:data dataSourceName="qcc-essbase" useAliases="true"
             query="\"All Time Periods\" <COLUMN (\"Scenario\") <SYM <ICHILD
             \"Scenario\" <ROW (\"All Products\") \"All Products\" !" />

       </blox:present>
       </body>
       </html>
```

Dial Chart Tag Reference

This section describes the tag attributes for the custom JSP tags supporting the creation of dial charts. The information is organized as follows:

<blox:dial> Tag Attributes

The following table lists the attributes for the <blox:dial> tag. For information on what a dial is and what its common properties are, see "Dial" on page 130.

Attribute	Default	Description
borderColor	black	The color of the border for the dial.

Attribute	Default	Description
borderType	solid	The border type for the border around the edge of the dial. Available values are: none and solid. For example: borderType="none"
color	default in the charting engine	The fill color of the dial chart. Specify a Java color or a hexadecimal value (for example, #CCCCFF).
formatMask	null	The customized format to use for the dial's labels. The format masks are set in the same way as "x1FormatMask" on page 115, "y1FormatMask" on page 121, and "y2FormatMask" on page 125. The keywords K and M (for thousands and millions) are supported. Division (/) and multiplication (*) are also supported. For example: formatMask="$###K"
radius	100	The radius of the dial as a percentage of the available space. Valid values are 0 through 100.
showLabels	true	To display the labels for the tics. If there is a scale for this dial, then the labels will be applied to the tics starting from the least value to the greatest value. Any missing labels or excess labels will be blank or ignored.
startAngle	-90	The position at which the dial region will begin. This angle is an upward pointing vertical line and sweeps in a clockwise direction. Values range from -180 to 180. For instance, having a starting angle of -90 and an ending angle of 90 will produce a horizontal half-circle dial.
stopAngle	90	The position at which the dial region will end. This angle is an upward pointing vertical line and sweeps in a clockwise direction. Values range from -180 to 180. For instance, having a starting angle of -90 and an ending angle of 90 will produce a horizontal half-circle dial.
ticPosition	outside	Where to position the tics for the dial axis. Possible values include: • inside: position the tics on the circumference of the dial pointing inward • outside: position the tics on the circumference of the dial pointing outward • none: leave off tics all together

<blox:needle> Tag Attributes

The following table lists the attributes for the <blox:needle> tag. For information on what a dial is and what its common properties are, see "Needle" on page 132. For an example, see "Example 2: Specifying Needles and Scope" on page 133.

Attribute	Default	Description
color	black	The color of the needle (both the needle and the needle end).

Attribute	Default	Description
endType	sharpArrow	Specifies what will be drawn for the end of the needle. Possible values are: • sharpArrow: a triangle with sharp back corners • blockArrow: a triangle at the end • round: a circle at the end of the needle • none: a plain line needle When using round needle end, endWidth should be set to larger than the needleWidth since a circle with a diameter of the same width as the needle will not be discernible.
endWidth	5	The width of the end of the needle in pixels. If endType is set to none, then endWidth is ignored
needleWidth	2	The width of the needle in pixels.
scope	The first data value	The cell whose value the needle should be pointing to. The scope should be specified as a series of dimension and member sets enclosed in braces. Use either display names or unique names. scope applies only to the row and column axes. If a dimension in the scope is not present, the scope will still match. For IBM DB2 OLAP Server and Hyperion Essbase data sources, specify the scope as follows: `scope="{d0:m0} {d1:m1} ..."` where d0 denotes a dimension and m0 denotes a member within that dimension. For example, for IBM DB2 OLAP Server and Hyperion Essbase data sources: `scope="{scenario:budget}"` For MS OLAP, Alphablox Cube Server, and SAP BW data sources, specify the scope using unique names starting with the cube name as follows: `scope="{[My Cube].[Measures]: [Measures].[Profit]}"`
tooltip	The associated member names and the value	The tooltip to display when end users hover over the needle's end.
value	none	The value the needle points to on the dial.

<blox:scale> Tag Attributes

The following table lists the attributes for the <blox:scale> tag. For information on what a scale is and how it is specified, see "Scale" on page 131.

Attribute	Default	Description
maximum	200%	The maximum value on the dial. You can specify an actual maximum value or a percentage (include the % sign). For example, if the data value is 100,000 and maximum is set to 150%, the maximum value on the scale will be 150,000. In cases where the data value exceeds the maximum value on the dial, the pointer will be pointing at the end of the scale.

Attribute	Default	Description
minimum	0	The minimum value on the dial. You can specify an actual minimum value, or a percentage (include the % sign). For example, if the data value is 100,000 and minimum is set to 50%, the minimum value on the scale will be 50,000. In cases where the data value is lower than the minimum value on the dial, the pointer will be pointing at the beginning of the scale.
scope	The first data value	The cell whose value should be used to determine the values on the scale. scope should be specified as a series of dimension and member sets enclosed in braces. Use either display names or unique names. scope applies only to the row and column axes. If a dimension in the scope is not present, the scope will still match. For IBM DB2 OLAP Server and Hyperion Essbase data sources, specify the scope as follows: `scope="{d0:m0} {d1:m1}...]"` where d0 denotes a dimension and m0 denotes a member within that dimension. For example, for IBM DB2 OLAP Server and Hyperion Essbase data sources: `scope="{scenario:budget}"` For MS OLAP, Alphablox Cube, and SAP BW data sources, specify the scope starting with the cube name as follows: `scope="{[My Cube].[Measures]: [Measures].[Profit]}"`
stepSize	Automatically determined by dividing the scale into five regions	The step size between tic marks on the dial.

<blox:sector> Tag Attributes

The following table lists the attributes for the <blox:sector> tag. For information on what a sector is and how to specify a sector, see "Sector" on page 131. For an example, see "Example 1: Specifying Sectors" on page 132.

Attribute	Default	Description
color		The color of the dial sector. Specify a Java color name or a hexadecimal value.
innerRadius	0	The inner radius of this dial sector as a percentage of the dial's radius. Valid values are 0 to 100.
outerRadius	100	The outer radius of this dial sector as a percentage of the dial's radius. Valid values are 0 to 100.

Attribute	Default	Description
scope	The first data value	The cell whose value should be used to determine the values in the sector. scope should be specified as a series of dimension and member sets enclosed in braces. Use either display names or unique names. scope applies only to the row and column axes. If a dimension in the scope is not present, the scope will still match. For IBM DB2 OLAP Server and Hyperion Essbase data sources, specify the scope as follows: `scope="{d0:m0} {d1:m1}...]"` where d0 denotes a dimension and m0 denotes a member within that dimension. For MS OLAP, Alphablox Cube, and SAP BW data sources, specify the scope starting with the cube name as follows: `scope="{[My Cube].[Measures]: [Measures].[Profit]}"`
startValue		The start value for this sector. This should be set based on the dial's scale. For example, if the dial's scale has a minimum value of 100 and a maximum value of 500, then a red sector start value of 300 and stop value of 500 will make the region on the dial between 300 and 500 red. This value should be between the dial's scale minimum and maximum.
stopValue		The stop value for this sector. This should be set based on the dial's scale. For example, if the dial's scale has a minimum value of 100 and a maximum value of 500, then a red sector start value of 300 and stop value of 500 will make the region on the dial between 300 and 500 red. This value should be between the dial's scale minimum and maximum.
tooltip		The text to display when the mouse hovers over the sector.

Chapter 8. CommentsBlox Tag Reference

This chapter contains reference material for CommentsBlox tag attributes. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

CommentsBlox Overview

CommentsBlox allows you to provide cell commenting (also known as cell annotations) functionality to your application. In addition, you can use CommentsBlox for general commenting that are not tied to any other Blox. For example, you can allow users to add comments to a site, an application, a report, or a Web page.

Comments are stored in a JDBC accessible relational database. Supported databases include IBM DB2® UDB, Sybase, Microsoft SQL Server, and Oracle. This data source needs to be defined to DB2 Alphablox. DB2 Alphablox provides a **Comments Management** page under the **Server** link in the DB2 Alphablox **Administration** tab that lets you specify the relational data source to use for storing comments. From that page, you can create "collections" (data tables) to store comments. For cell-level comments, you will need to specify the multidimensional data source used in your GridBlox, the cube to use (for Microsoft Analysis Services), and the dimensions to include. For general comments, you only need to specify the name.

Note: If you are using DB2 UDB on z/OS systems for storing comments, you must select the ″IBM DB2 JDBC Type 4 Driver for z/OS″ data adapter when defining this data source to DB2 Alphablox. If you choose the ″IBM DB2 JDBC Type 4 Driver″ driver, you will not be able to create a comments collection.

User Interface

When the commentary functionality is set up and enabled on a GridBlox user interface, A Comments menu item becomes available from the right-click menu. A comment indicator appears in the corner of the cells that has comments.

CommentsBlox is a container for comments that share the same set of fields and the same address scheme. For cell-level comments, the comments have an addressing scheme that incorporates the subset of dimensions and cube information needed to identify the cell. For general comments not tied to data cells, the addressing scheme is simply a string that contains the name of the comment collection. These comments are called "named comments." Each CommentsBlox can have multiple named comment sets.

When users choose to display comments for a cell, the popped up window shows the address of the cell and all comments made on that cell, and the author and the time the comment was added. Only the author of the comment can delete the comment.

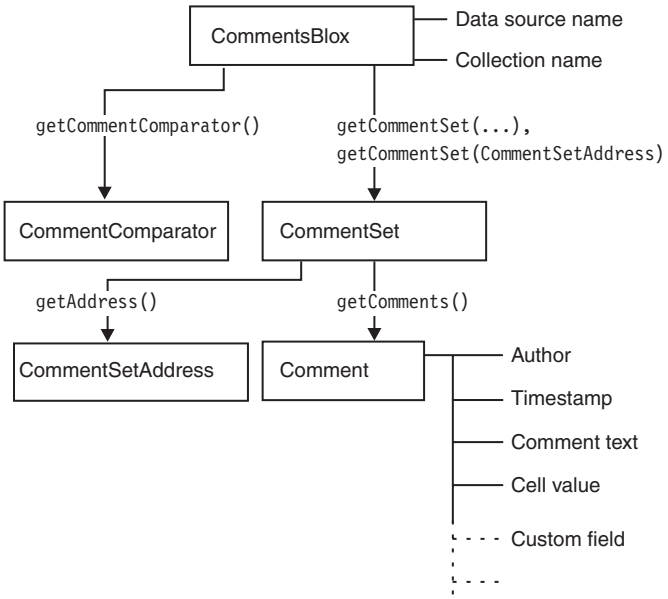
By default the comments are sorted by dates. Users can click the column headers to sort the comments based on the values in that column. Sort order works in toggle mode. Application developers can specify the field to sort on and the sort order using tags.

CommentsBlox Object Hierarchy and API

Through CommentsBlox, you can access a subset of comments associated with a specific cell or a named comment set, and then subsequently set or get more information on individual comment, such as its author, the comment text, and the time the comment was added.

Each comment has four required fields: author, timestamp, cell value, and comment text. You can add your own custom fields when creating the comments collection through the Comments Management page.

The following diagram shows the object hierarchy of CommentsBlox:



The CommentSet object is an interface through which comments are added, updated, and removed from the collection. Each CommentSet also has an address. As described earlier, for cell-level comments, the address of a CommentSet consists of the dimensions and cube information needed to identify the cell. For general comments not tied to data cells, the addressing scheme is simply a string that contains the name of the comment collection. You can access the CommentSet containing comments saved on a cell in a named address through CommentsBlox.

The Comment object has the following static fields that store the information for each comment:

- `FIELD_AUTHOR`
- `FIELD_CELLVALUE`
- `FIELD_COMMENTTEXT`

- `FIELD_TIMESTAMP`

It may also contain other custom fields as defined when the comment collection is created.

The Javadoc documentation for the Comment, CommentSet, CommentSetAddress, and CommentComparator objects are under the `com.alphablox.blox.comments` package.

CommentsBlox Events

CommentsBlox uses CommentsListener to notify an assigned CommentEvent listener (CommentAddedEvent, CommentDeletedEvent, and CommentUpdatedEvent) that a comment was changed in the comments collection. This allows you to perform custom actions such as logging comment changes *after* the events are processed by the server.

Using the CommentsBlox `addCommentsListener()` method, you can add your comment listener. The CommentsListener has a `CommentChanged()` method that lets you specify the comment event to listen for. Each of the CommentAddedEvent, CommentDeletedEvent, and CommentUpdatedEvent lets you access the Comment or CommentSet affected by the associated event. For an example, see "Example 4: Adding a CommentAddedEvent Listener" on page 146.

Database Operations and Permissions

The use of CommentsBlox involves various database operations to support the creation of comments collections, editing a collection, and adding, displaying, and deleting comments. The following table shows the data operations behind the scene depending on the tasks performed. This will help you plan for proper permission setup needed for your application to work.

Task Performed	Data Operations Involved
Create a Comments collection:	Create tables and indexes
Edit an existing Comments collection:	Drop the old tables and create new ones
Delete a Comments collection:	Delete the associated tables
Add a comment:	Update and insert
Delete a comment:	Delete
Display a comment	Select

CommentsBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to use the custom tag to create a CommentsBlox. For a copy and paste version of the tag with all the attributes, see "CommentsBlox JSP Custom Tag" on page 460. Note that the CommentsBlox tag is a nested tag within the DataBlox custom tag when you want to provide cell-level comments. For named comments, since the comments are not tied to a DataBlox, the CommentsBlox tag is used as a standalone tag.

Syntax

For cell-level comments (associated with a DataBlox):

```
<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>"
    ... >
    <blox:comments
        [attribute="value"] >
       <blox:sortComments
          field="" order="" />
    </blox:comments>
</blox:data>
```

Or you can have a standalone CommentsBlox referenced in a DataBlox:

```
<blox:comments id="myComments"
    [attribute="value"] >
       <blox:sortComments
          field="" order="" />
</blox:comments>

<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>"
    ... >
    <blox:comments
        bloxRef="myComments">
    </blox:comments>
</blox:data>
```

Note: You cannot add a CommentsBlox tag within a DataBlox tag that is using the `bloxRef` attribute. It has to be nested within an actual DataBlox tag where its `dataSourceName` and `query` attributes are defined.

For named comments (not associated with a DataBlox):

```
<blox:comments
    [attribute="value"] >
       <blox:sortComments
          field="" order="" />
</blox:comments>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
bloxName
bloxRef
commentsOnBaseMember
collectionName
dataSourceName
password
userName

The nested <blox:sortComments> tag is optional and has two attributes:

Attribute
field
sort

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing </blox:comments> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
dataSourceName = "comments_mssql" />
```

Examples

For cell-level comments (associated with a DataBlox):

```
<blox:data id = "myData1"
    dataSourceName = "foodmart"
    query = " <%=myQueryString %>" >
    <blox:comments id = "myComments1"
        collectionName = "sales_comments"
        dataSourceName = "comments_mssql" />
</blox:data>
```

For named comments (not associated with a DataBlox):

```
<blox:comments id = "myComments1"
    collectionName = "sales_comments"
    dataSourceName = "comments_mssql" />
```

Cell-level comments with a specified field to sort on when the Display Comments window pops up:

```
<!--import the following package in order to use the field constants-->
<%@ page import="com.alphablox.blox.comments.*" %>

<blox:data id = "myData2"
    dataSourceName = "QCC-MSAS"
    query = " <%=myQueryString %>" >
    <blox:comments id = "myComments2"
        collectionName = "planning_comments"
        dataSourceName = "comments_mssql" >
        <blox:sortComments
            field="<%= Comment.FIELD_COMMENTTEXT %>"
            order="<%= CommentComparator.DESCENDING %>" />
    </blox:comments>
</blox:data>
```

CommentsBlox Examples

This section provides examples that demonstrate how to use CommentsBlox to enable cell-level comments (associated with a DataBlox and a GridBlox), how to access comments for an individual cell through the MDBResultSet, and how to add an event listener when a comment is added (or deleted or updated).

- Example 1: Enabling cell commenting
- Example 2: Specifying Field to Sort On and Sort Order
- Example 3: Accessing Cell Comments Using MDBResultSet

- Example 4: Adding a CommentAddedEvent Listener

For page-level comments (not associated with a DataBlox) or using custom pages for adding and displaying cell comments, see the Highlighting and Commenting Data chapter in the *Developer's Guide*. For complete, live examples of CommentsBlox, see the Commenting on Data section in Blox Sampler under the Assembly tab on DB2 Alphablox Home Page.

Example 1: Enabling cell commenting

This example demonstrates how to enable cell commenting by setting `commentsEnabled` attribute to `true` on the GridBlox and adding a nested CommentsBlox tag inside the DataBlox. Note that the relational data source used to store comments needs to have been defined to DB2 Alphablox, and the collection name needs to have been created via the Comments Management page under the DB2 Alphablox Administrative tab.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    String query = "your_data_query_here";
%>

<html>
<head>
    <blox:header />
</head>
<body>
    <blox:present id="presentBlox">
        //Enable cell commenting UI in GridBlox
        <blox:grid
            commentsEnabled="true" />

        <blox:data
            dataSourceName="foodmart"
            query="<%=query%>">
            //The datasource and collection names are defined and
            //and created via the DB2 Alphablox Admin Pages
            <blox:comments
                collectionName="sales_comments"
                dataSourceName="comments_mssql" >
            </blox:comments>
        </blox:data>
    </blox:present>
</body>
</html>
```

Example 2: Specifying Field to Sort On and Sort Order

This example is the same as the previous example except that it specifies the default field to sort the comments on when users choose to display comments on a cell. The sort order is set to ascending. The `com.alphablox.blox.comments` package is imported in order to use the constants for the field names and sort order.

```
<%@ page import="com.alphablox.blox.comments.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%
    String query = "your_data_query_here";
%>
<html>
<head>
    <blox:header />
</head>
<body>
    <blox:present id="presentBlox" mayscriptEnabled="true" >
        //Enable cell commenting UI in GridBlox
```

```
        <blox:grid
            commentsEnabled="true" />
        <blox:data
            dataSourceName="foodmart"
            query="<%=query%>">
            //The datasource and collection names are defined and
            //and created via the DB2 Alphablox Admin Pages
            <blox:comments
                collectionName="sales_comments"
                dataSourceName="comments_mssql" >
                <blox:sortComments
                   field="<%= Comment.FIELD_AUTHOR %>"
                   order="<%= CommentComparator.ASCENDING %>" />
            </blox:comments>
        </blox:data>
    </blox:present>
</body>
</html>
```

Example 3: Accessing Cell Comments Using MDBResultSet

This example demonstrates how to access individual comments associated with a cell through the MDBResultSet object. To access the comments associated with Truffles for FY2001 in the following GridBlox:

Product (Category)	FY2000	FY2001	FY2002
Chocolate Blocks	178,148	3,282,371	3,052,920
Chocolate Nuts		6,686,802	9,390,529
Specialties	295,497	7,769,125	7,909,836
Truffles		749,789	789,908
All Products	473,645	18,488,088	21,143,193

you would access the MDBResultSet of the underlying DataBlox and then get to cell(1,3):

```
MDBResultSet resultSet = (MDBResultSet)
myCommentGrid.getDataBlox().getResultSet();
Cells cells = resultSet.getCells();
Cell cell = cells.getCell(1,3);
```

Now you can get to all comments for that cell:

```
CommentSet truffleCommentSet = cell.getCommentSet();
```

Here is the complete code:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="com.alphablox.blox.data.mdb.*" %>
<%@ page import="com.alphablox.blox.*" %>
<%@ page import="com.alphablox.blox.comments.*" %>
<%@ taglib uri="bloxtld" prefix="blox" %>
<html>
<head>
   <blox:header/>
</head>
<body>
<blox:grid id="myCommentGrid"
   width="60%"
   height="50%"
   commentsEnabled="true"
   defaultCellFormat="#,###"
   bandingEnabled="true">
   <blox:data dataSourceName="QCC-MSAS"
```

```
      query="SELECT {[Time.Fiscal].[All Time Periods].Children} ON COLUMNS,
         {[Products.Category].[All Products].Children,
         [Products.Category].[All Products]} ON ROWS FROM QCC
          WHERE ([Measures].[Sales])">
      <blox:comments
         collectionName="CommentsCollectionMSAS"
         dataSourceName="CommentsCollectionMSAS" />
   </blox:data>
</blox:grid>

<%
   //Access the comments associated with a cell from the result set
   MDBResultSet resultSet = (MDBResultSet)
myCommentGrid.getDataBlox().getResultSet();
   Cells cells = resultSet.getCells();
   Cell cell = cells.getCell(1,3);
   CommentSet truffleCommentSet = cell.getCommentSet();

   //Now get the address of the CommentSet for this cell
   CommentSetAddress truffleAddress = truffleCommentSet.getAddress();
   out.write("<BR>Address of CommentSet for Truffles: "+truffleAddress +
"<br>");

   //Now get the comment text for each comment in the CommentSet
   Comment[] comments = truffleCommentSet.getComments();
   out.write("<BR>The number of comments is: "+comments.length);
   for(int i = 0; i < comments.length; i++) {
      out.write("<BR>Comment Text: "+comments[i].getCommentText()+" for
comment: "+ i + "<br>");
   }

%>
</body>
</html>
```

The output looks as follows:

```
Address of CommentSet for Truffles: CellCommentAddress: [Locations]:[Locations].[All
Locations];[Measures]:[Measures].[Sales];[Products].[Category]:[Products].[Category].[All
Products].[Truffles];[Products].[Code]:[Products].[Code].[All
Products];[Products].[Seasonal]:[Products].[Seasonal].[All Products];[Scenario]:[All
Scenario];[Seasonal]:[Seasonal].[All Seasonal];[Time].[Calendar]:[Time].[Calendar].[All Time
Periods];[Time].[Fiscal]:[Time].[Fiscal].[All Time Periods].[FY2001];

The number of comments is: 2

Comment Text for comment 0: The sales in the East region were 32% higher than projected,
making up the lost of sales in the West due to machine breakdown.

Comment Text for comment 1: There was a machine breakdown in the west region for two
weeks that impacted the sales of seasonal items.
```

Example 4: Adding a CommentAddedEvent Listener

The following example demonstrates how to capture a CommentAddedEvent and then print the author, comment text, and timestamp to the Alphablox console. To add a comment event listener:

1. Use the CommentsBlox `addCommentsListener()` method to add your comment listener.
2. Your comment listener should implement the CommentsListener interface.

3. Add the action to take when the comment is changed. Specify either the CommentAddedEvent, CommentDeletedEvent, or CommentUpdatedEvent to listen to in the `CommentChanged()` method.
4. You can then use `getComment()` or `getCommentSet()` to access the associated Comment or CommentSet.

```
<%@ taglib uri = "bloxtld" prefix = "blox"%>
<%@ page import="com.alphablox.blox.comments.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.*" %>
<blox:comments id="myComments"
    collectionName="CommentsCollection"
    dataSourceName="CommentsCollection" />

<%! public abstract class CListener implements CommentsListener
   {
   BloxModel model;
   public void commentsChanged(com.alphablox.blox.comments.CommentAddedEvent cadded)
     throws Exception
     {
        Comment comment = cadded.getComment();
        StringBuffer msg = new StringBuffer("---------" + "\n");
        msg.append("Author: " + comment.getAuthor() + "\n");
        msg.append("Comment text: " + comment.getCommentText() + "\n");
        msg.append("Time: " + comment.getTimestamp( ));
        MessageBox msgBox = new MessageBox(msg.toString(),"Comments Added",
           MessageBox.MESSAGE_OK, null);
        model.getDispatcher().showDialog(msgBox);
     }
  } %>
  <blox:present id="CommentsPresentBlox"
    ...
     >
    <blox:grid
       commentsEnabled="true" />
       <blox:data
          dataSourceName="QCC-Essbase"
          query="!">
          <blox:comments
             bloxRef="myComments"/>
       </blox:data>
       <% myComments.addCommentsListener( CListener() );  %>
 </blox:present>
```

CommentsBlox Tag Attributes

The following are unique CommentsBlox tag attributes:

- commentsOnBaseMember
- collectionName
- dataSourceName
- userName

id

This is a common Blox tag attribute. For a complete description, see “id” on page 36.

bloxName

This is a common Blox tag attribute. For a complete description, see “bloxName” on page 32.

bloxRef

This is a common Blox tag attribute. For a complete description, see “bloxRef” on page 35

commentsOnBaseMember

Specifies whether comments created on shared members are associated with the base member or an instance of the shared member. This allows comments to be displayed with both the base member and all the shared versions of that member or with a particular instance of a shared member.

Data Sources

DB2 OLAP Server and Hyperion Essbase

Syntax

`commentsOnBaseMember="`*`flag`*`"`

where:

Argument	Default	Description
`flag`	`false`	When this property is set to `false` (the default), a comment placed on a shared member will only appear on that particular instance of the shared member. When `true`, comments created on a shared member are associated with the base member so they are displayed on both the base member and all instances of the shared member.

Usage

This flag for `commentsOnBaseMember` is used at the time the comment is created, not at the time it is displayed. You can change the value of the flag at runtime using the associated Java method to cause comments to be associated specifically with the given instance of a shared member (`setCommentsOnBaseMember(false)`) or with the base member (`setCommentsOnBaseMember(true)`). This property affects only DB2 OLAP Server and Hyperion Essbase data sources.

collectionName

The name of the comment collection. Each CommentsBlox needs to be associated with a relational data source and a collection name in that data source.

Data Sources

Relational (for storing comments)

Syntax

`collectionName="`*`name`*`"`

where:

Argument	Default	Description
`name`	`null`	The name of the comment collection.

dataSourceName

The name of the data source used to stored comments.

Data Sources
Relational (for storing comments)

Syntax
`dataSourceName="name"`

where:

Argument	Default	Description
`name`	`null`	The name of the data source.

password
The password to use to connect to the data source used for storing comments.

Data Sources
Relational (for storing comments)

Syntax
`password="password"`

where:

Argument	Default	Description
`password`	The password specified in the data source definition via the DB2 Alphablox Admin Pages.	The password to use to connect to the data source

userName
The username to use to connect to the data source used for storing comments.

Data Sources
Relational (for storing comments)

Syntax
`userName="username"`

where:

Argument	Default	Description
`username`	The username specified in the data source definition via the DB2 Alphablox Admin Pages.	The username to use to connect to the data source

Usage
When the collection is first created, this username must have sufficient privileges to create tables and indexes. For retrieving and writing comments, this user must have `connect` privileges to the database, and `select` privilege if the user will only be reading comments. The `insert` and `update` privileges are required if the user will be adding or modifying comments.

Chapter 9. ContainerBlox Tag Reference

This chapter contains reference material for ContainerBlox properties, methods and objects. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

ContainerBlox Overview

ContainerBlox is an empty Blox with no pre-defined behaviors. DataBlox and all presentation Blox (PresentBlox, GridBlox, ChartBlox, ToolbarBlox, PageBlox, and DataLayoutBlox) are extensions of ViewBlox, which is an extension of ContainerBlox. ContainerBlox allows you to create an area on the page to utilize any user interface components the DB2 Alphablox provides, such as menus and buttons.

You can create your own custom Blox by extending the ContainerBlox to take advantage of the services it provides, such as no page refreshes, full server-side control and logic, use of all core components (for example, the Tree control, menus, and toolbars), same user interface programming model, easy distribution, localizable resources files, and dialogs. Java developers can extend the ContainerBlox to add controls within a Blox, such as adding a drop-down list to the DataLayoutBlox. They can also create reusable, self-contained extensions to the built-in Blox. For example, Java developers may add a color picker dialog to allow users to assign colors to pie slices in a pie chart by utilizing the existing ChartBlox color properties.

ContainerBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each blox. This section describes how to create the custom tag to create a ContainerBlox. For a copy and paste version of the tag with all the attributes, see "ContainerBlox JSP Custom Tag" on page 461.

Syntax

```
<blox:container
    [attribute="value"] >
</blox:container>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
bloxName
enablePoppedOut

Attribute
height
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
render
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

Examples

```
<blox:container id="myContainer"
    width="200"
    height="100"
/>
```

ContainerBlox Tag Attributes

This section describes the tag attributes supported by ContainerBlox alphabetically. For ContainerBlox methods see the ContainerBlox class in the com.alphablox.blox package in the Javadoc documentation. Since most Blox inherits from ContainerBlox, the tag attributes listed here can also be found in "Tag Attributes Common to Multiple Blox" on page 30.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 36.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 32.

enablePoppedOut

Specifies whether a Blox can be opened in a separate window; that is, "popped out" of the application page.

Data Sources

All

Syntax

enablePoppedOut="*enablePoppedOut*"

where:

Argument	Default	Description
enablePoppedOut	true	A boolean argument. A value of true specifies that a user can pop out a blox to another window, a value of false disables this feature.

Usage
By default, enablePoppedOut is set to true, and users can have the Blox display in a popped out browser window by clicking the Popped Out button in the toolbar or selecting the "Popped Out" option from the View menu. When enablePoppedOut is set to false, this button and menu option are disabled. To remove the button and the menu item, use the Blox UI tags to remove the UI component. See "Menubar, Menu, and MenuItem" on page 393.

poppedOut and its related properties apply to PresentBlox and standalone GridBlox/ChartBlox.

Examples
```
<blox:present id="myPresentBlox"
   enablePoppedOut="false"
   ...>
   ...
</blox:present>
```

See Also
"poppedOut" on page 153, "poppedOutHeight" on page 154, "poppedOutTitle" on page 154, "poppedOutWidth" on page 155

height
This is a common Blox property. For a complete description, see "height" on page 35.

poppedOut
Specifies whether the Blox is to display in a separate window, or "popped out" of the application page when the Blox is loaded.

Data Sources
All

Syntax
poppedOut="*popOut*"

where:

Argument	Default	Description
popOut	false	A boolean argument. A value of true specifies that the Blox displays in a popped out window when the Blox is loaded. A value of false indicates that the Blox displays in the same window as the page.

Usage
Applies to a PresentBlox, a standalone GridBlox, or a standalone ChartBlox.

Examples

```
<blox:present id="myPresentBlox"
   poppedOut="true"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also

"enablePoppedOut" on page 152, "poppedOutHeight" on page 154, "poppedOutTitle" on page 154, "poppedOutWidth" on page 155

poppedOutHeight

Specifies the height (in pixels) of the Blox in the separate, or popped out, window.

Data Sources

All

Syntax

```
poppedOutHeight="newHeight"
```

where:

Argument	Default	Description
newHeight	600	Integer specifying the popped-out window height in pixels.

Examples

```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also

"enablePoppedOut" on page 152, "poppedOut" on page 153, "poppedOutTitle" on page 154, "poppedOutWidth" on page 155

poppedOutTitle

Specifies the title of the separate, or popped out, window in which the Blox is displayed.

Data Sources

All

Syntax

```
poppedOutTitle="title"
```

where:

Argument	Default	Description
title	Popout Blox Window	Any string. The specified string is displayed as the title of the popped out window.

Usage
The default displays the name of the applet as the window title.

Examples
```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also
"enablePoppedOut" on page 152, "poppedOut" on page 153, "poppedOutHeight" on page 154, "poppedOutWidth" on page 155

poppedOutWidth
Specifies the width, in pixels, of the Blox in the separate, or popped out, window.

Data Sources
All

Syntax
```
poppedOutWidth="newWidth"
```

where:

Argument	Default	Description
newWidth	800	Integer specifying the popped-out window width in pixels.

Usage
The setPoppedOutWidth method has no effect if the Blox is already popped out.

Examples
```
<blox:present id="myPresentBlox"
   poppedOutHeight="800"
   poppedOUtWidth="1000"
   poppedOutTitle="Sales Analysis Window"
   ...>
   ...
</blox:present>
```

See Also
"enablePoppedOut" on page 152, "poppedOut" on page 153, "poppedOutHeight" on page 154, "poppedOutTitle" on page 154

render

This is a common Blox property. For a complete description, see “render” on page 38.

visible

This is a common Blox property. For a complete description, see “visible” on page 40.

width

This is a common Blox property. For a complete description, see “width” on page 40.

Chapter 10. DataBlox Tag Reference

This chapter contains reference material for DataBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

DataBlox Overview

DataBlox offers the following functionality:

- provides a representation of a data set (in grid form), either relational or multidimensional, for the assembler to access
- enables application scripting (such as executing a query)
- serves as a data source for other Blox (such as ChartBlox or GridBlox)

DataBlox not only provides a means to access and query data, but also returns a ResultSet object and a MetaData object. The result set returned involves actual data values from a query and enables you to perform tasks such calculations or custom data view. The metadata object contains information on the cubes, dimensions, and members (outline) of the data source. For more information on the DataBlox object model, see "DataBlox—Access to Metadata and Result Sets" on page 7.

To use the APIs associated with RDBMetaData and RDBResultSet, you need to import the com.alphablox.blox.data.rdb package in your JSP page:

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
```

To use the APIs associated with MDBMetaData and MDBResultSet, you need to import the com.alphablox.blox.data.mdb package in your JSP page:

```
<%@ page import="com.alphablox.blox.data.mdb.*" %>
```

DataBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a DataBlox. For a copy and paste version of the tag with all the attributes, see "DataBlox JSP Custom Tag" on page 461.

Syntax

```
<blox:data
      [attribute="value"] >
</blox:data>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
bloxName
bloxRef
aliasTable
applyPropertiesAfterBookmark
autoConnect
autoDisconnect
bookmarkFilter
calculatedMembers
catalog
columnSort
connectOnStartup
credential
dataSourceName
dimensionRoot
drillDownOption
drillKeepSelectedMember
drillRemoveUnselectedMembers
enableKeepRemove
enableShowHide
hiddenMembers
hiddenTuples
leafDrillDownAvailable
memberNameRemovePrefix
memberNameRemoveSuffix
mergedDimensions
mergedHeaders
onErrorClearResultSet
parentFirst
password
performInAllGroups
provider
query
retainSlicerMemberSet
rowSort
schema
selectableSlicerDimensions
showSuppressDataDialog
suppressDuplicates
suppressMissingColumns
suppressMissingRows

Attribute
suppressNoAccess
suppressZeros
textualQueryEnabled
useAASUserAuthorizationEnabled
useAliases
useOlapDrillOptimization
userName

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:data>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
useAliases="true" />
```

Examples

```
<blox:data
     dataSourceName="QCC-Essbase"
     query="<ROW (\"All Products\") <ICHILD \"All Products\"
             <COLUMN (\"All Time Periods\") <CHILD \"All Time Periods\"
             <PAGE (Measures) Sales !"
     useAliases="true"
     selectableSlicerDimensions="All Locations" >
</blox:data>
```

When a `data` tag is either nested within the tag of a data presentation Blox (PresentBlox, GridBlox, ChartBlox, DataLayoutBlox, PageBlox, or MemberFilterBlox), it cannot have an `id`. A common practice is to define a standalone DataBlox with an `id` and later reference it in your presentation Blox as follows:

```
<blox:data id="myDataBlox"
     dataSourceName="QCC-Essbase"
     query="<ROW (\"All Products\") <ICHILD \"All Products\"
             <COLUMN (\"All Time Periods\") <CHILD \"All Time Periods\"
             <PAGE (Measures) Sales !"
     useAliases="true"
     selectableSlicerDimensions="All Locations" >
</blox:data>

<blox:present id="myPresentBlox" >
     <blox:data bloxRef="myDataBlox" />
     ...
</blox:present>
```

This allows you to use the same DataBlox in multiple presentation Blox for synchronized views of the same data, or to access and change the DataBlox property directly using the DataBlox `id` in Java scriptlets.

DataBlox Tag Attributes by Category

This section lists the tag attributes unique to GridBlox. For tag attributes common to several Blox, see "Common Blox Tag Attributes by Category" on page 29. The tag attributes supported by DataBlox are organized as follows:

For DataBlox methods, see the DataBlox class in the com.alphablox.blox package in the Javadoc documentation.

Data Appearance

The following DataBlox tag attributes are associated with the appearance of the data:

- `memberNameRemovePrefix`
- `memberNameRemoveSuffix`
- `mergedDimensions`
- `mergedHeaders`
- `showSuppressDataDialog`
- `suppressDuplicates`
- `suppressMissingColumns`
- `suppressMissingRows`
- `suppressNoAccess`
- `suppressZeros`
- `textualQueryEnabled`
- `useOlapDrillOptimization`

Data Source

The following are DataBlox tag attributes associated with data sources:

- `aliasTable`
- `autoConnect`
- `autoDisconnect`
- `catalog`
- `credential`
- `dataSourceName`
- `onErrorClearResultSet`
- `password`
- `provider`
- `query`
- `schema`
- `useAASUserAuthorizationEnabled`
- `userName`

Data Manipulation

The following are DataBlox tag attributes used to manipulate data:

- `calculatedMembers`
- `columnSort`

- dimensionRoot
- drillDownOption
- drillKeepSelectedMember
- drillRemoveUnselectedMembers
- enableKeepRemove
- enableShowHide
- hiddenMembers
- hiddenTuples
- internalSortEnabled
- leafDrillDownAvailable
- parentFirst
- performInAllGroups
- retainSlicerMemberSet
- rowSort
- selectableSlicerDimensions
- useAliases

DataBlox Tag Attributes

This section describes tag attributes supported by DataBlox in alphabetical order. For DataBlox methods, see the DataBlox class in the com.alphablox.blox package. For complete descriptions of common Blox tag attributes, see "Tag Attributes Common to Multiple Blox" on page 30.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 36.

bloxName

This is a common Blox tag attribute. For a complete description, see "bloxName" on page 32.

bloxRef

This is a common Blox tag attribute. For a complete description, see "bloxRef" on page 35.

aliasTable

Specifies the IBM DB2 OLAP Server or Hyperion Essbase alias table to use with the data source.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase

Syntax

aliasTable="*aliasTable*"

where:

Argument	Default	Description
aliasTable	empty string	Name of an IBM DB2 OLAP Server or Hyperion Essbase alias table

Usage

The value for aliasTable is one of the values provided when defining a data source to DB2 Alphablox. If no aliasTable property is specified on a DataBlox, the value is taken from the corresponding data source definition, if the value is present.

The alias tables in the IBM DB2 OLAP Server or Hyperion Essbase database must have names containing only ASCII characters.

See Also

"dataSourceName" on page 183

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see "applyPropertiesAfterBookmark" on page 30.

autoConnect

Allows the DataBlox to connect to the database automatically whenever it needs database access.

Data Sources

Relational Only

Syntax

autoConnect="*autoConnect*"

where:

Argument	Default	Description
autoConnect	false	Specify true to enable autoConnect; false to disable it.

Usage

The autoConnect property allows a DataBlox to reconnect to the data source automatically when the application requires a connection. It will not connect to the data source on startup; use the DataBlox connect() method or the connectOnStartup for this.

You can use autoConnect and autoDisconnect together to create an application where connections are opened only when needed and closed once the requested data is retrieved, without erasing the result set. When autoConnect is enabled and the user performs an operation that requires a data source connection, the DataBlox connects to the data source, restores the current query, and then executes the operation. If autoDisconnect is also enabled, DataBlox disconnects from the data source when the operation is complete.

Important: Connecting to a data source and restoring a query are time-intensive processes. The `autoConnect` and autoDisconnect properties should be used only in certain cases, such as when the number of connections to a database is limited.

If `autoConnect` is disabled and autoDisconnect is enabled, the user will not be able to perform operations on the result set after the initial disconnect.

See Also

"autoDisconnect" on page 163, "connectOnStartup" on page 181, `connect()`, `disconnect()`

autoDisconnect

Allows the DataBlox to disconnect from the database automatically whenever a data access operation is complete.

Data Sources

Relational; Microsoft Analysis Services; SAP BW

Syntax

`autoDisconnect="`*`autoDisconnect`*`"`

where:

Argument	Default	Description
`autoDisconnect`	`false`	Specify `true` to enable `autoDisconnect`; `false` to disable it.

Usage

The `autoDisconnect` property allows a DataBlox to disconnect from the data source automatically when the application no longer requires a connection, without erasing the current result set. If the `connect()` method is set to have the DataBlox connect on startup (or the connectOnStartup property is set to `true`) and `autoDisconnect` is enabled, the DataBlox will connect, restore a query if applicable, and then disconnect.

You can use `autoDisconnect` and autoConnect together to create an application where connections are opened only when needed and closed once the requested data is retrieved. When `autoDisconnect` is enabled, the DataBlox disconnects from the data source whenever an operation requiring a connection is complete. If autoConnect is also enabled, the DataBlox will automatically reconnect to the data source when necessary.

Important: Connecting to a data source and restoring a query are time intensive processes. The autoConnect and `autoDisconnect` properties should be used only in certain cases, such as when you are experiencing scalability problems with Microsoft Analysis Services due to large client cache memory consumption per connection. In this case, if you have custom code that performs metadata operations such as a for loop with thousands of `resolveMember()` calls, you should call the `clearClientCache()` method afterwards to free up the memory. See the Connecting to Data chapter in the *Developer's Guide*.

The autoDisconnect property does not apply to the RDB metadata object on a server-side DataBlox. The DataBlox will not disconnect from an RDB data source after a metadata request, even if autoDisconnect is set to true. The application must explicitly disconnect when it is through with the object. However, you can still use autoConnect to reconnect to the data source on future metadata requests.

If autoDisconnect is enabled and autoConnect is disabled, the user will not be able to perform operations on the result set after the initial disconnect.

See Also

"autoConnect" on page 162, "connectOnStartup" on page 181, connect(), disconnect()

bookmarkFilter

This is a common Blox property. For a complete description, see "bookmarkFilter" on page 31.

calculatedMembers

Specifies a new member that is calculated by DB2 Alphablox using the result set retrieved from the data source. Members used in calculation have to exist in the result set or the calculated member will not show.

Data Sources

All

Syntax

calculatedMembers="*definitionString*"

where:

Argument	Default	Description
definitionString	empty string	A comma-delimited list of one or more calculated member definitions. Specify each definition as shown below.

Specify each definition within the definitionString as follows:

dim:*calc*{*refMember*:*gen:*missingIsZero:*drillDownMember*:*drillUpMember*}=*expression*{*scopeDim*:*scopeMember*}

where:

definitionString Component	Description
dim	The name of the dimension on which to create a calculated member. **Note:** If you are merging headers using the mergedHeaders property, you should still use the original dimension name rather than the merged dimension header since calculatedMembers is performed before the dimension headers are merged.

definitionString Component	**Description**
`calc`	The name of calculated member. The name could be the same as an existing member or dimension name since a calculated member has an internal unique name in the form `ABXCalc_`*dimName_calc*`.` For example, if a "Total" calculated member is added to the Product dimension, it has a unique name of ABXCalc_Product_Total. This is used internally and the only time it is displayed to the user is when the user turns on unique names in the Dimension Explorer. It never shows in the grid, even when the "use aliases" data option is turned off.
`refMember`	The name of an existing member, including other calculated members, before which this calculated member is to be placed. Specifying the reference member is optional. You must place double quotes around member names that contain special characters. For example: `calculatedMembers="Product:\"Profit %\"{missingIsZero} = Gross Margin/Sales*100"` The calculated member *calc* will be placed before *refMember* in the grid. If you do not specify a reference member, the calculated member *calc* will be placed in the last row or column. If the user drills on or hides the reference member, the calculated member retains its position. However, if the user removes the reference member, the calculated member moves to the last row or column. See "Example 2: Specifying the position of the calculated member" on page 175.
`gen`	The generation number of the calculated member. Specifying a generation number is optional. Generation can be defined as either absolute or relative to the `refMember`, and it determines the indentation of the calculated member on the axis. The default generation number is 1. To specify absolute generation, define *gen* as a positive integer. A colon is required before the generation number even if there is no reference member defined. To specify relative generation, define *gen* as a + (plus) or - (minus) operator followed by an integer. The calculated member's generation will be the reference member's generation number plus or minus the integer defined in `gen`. A reference member must be defined in order to use relative generation, and the two must be separated by a colon.
`missingIsZero`	Optional keyword (case-insensitive) to use if you want all missing values for members involved in the calculation to be treated as zero. By default, all missing values in the calculation are treated as missing. To change the default behavior, use this special keyword. The following example will treat all missing values in Product1, Product2, and Product3 as zero. `Product:Total Sales{missingIsZero} = Product1 + Product2 + Product3` **Note:** This keyword only affects calculations using member variables. It has no effect on calculation functions.

definitionString Component	**Description**
`drillDownMember`	Optional. The real member to drill down when users try to drill down on this calculated member.
`drillUpMember`	Optional. The real member to drill up when users try to drill up from this calculated member. For example: `Year:Total{:::Qtr3:Jul} = sum()` This would drill down on Qtr3 and drill up on Jul from the calculated member "Total." Notice that when the optional *refMember*, *gen*, and `MissingIsZero` are not specified inside the curly braces, you should include the colons so the *drillDownMember* will not be taken as the *refMember*.
`expression`	The arithmetic expression involving members of *dim* or values from other dimensions. For example, `calculatedMembers="All Products:Products 1 and 2 = Product1 + Product2"` adds a calculated member called "Products 1 and 2" in the "All Products" dimension. This expression involves only members from the same dimension where the calculated member is added. If the calculation involves values from the intersection of multiple dimensions, you should specify the dimension where each member is from by separating the dimension and member name with a colon, and then separate each *dimesnion:member* pair with a semi-colon: *dim1:member1;dim2:member2;...;dimN:memberN* where: • at least one dimension is from the row axis • at least one dimension is from the column axis • for each semicolon delimited *dimension:member* pair, the dimension name has to be supplied, followed by a colon and the member of that dimension In the following example, a calculated member "Percentage of Total" is added to the All Products dimension with values from the intersection of All Products and All Locations as dividers: `calculatedMembers="All Products: Percentage of Total= All Products / All Locations:All Locations; All Products:All Products` See "Example 5: Calculations involving members from different dimensions" on page 176 for more details. For functions supported in calculations, see "Functions for CalculatedMembers" on page 168 for a detailed listing and description.

definitionString Component	**Description**
`scopeDim: scopeMember`	Defines the dimension and members for which the calculated member is displayed. Additional scope members are separated by commas. Additional pairs are divided within the braces by a semicolon. Specifying a scope is optional. Member names that contain special characters should be enclosed in double quotes (note that you need to escape inner double quotes). When a scope is defined, the calculated member appears only for the members specified in the scope. However, if the calculated member and a scope member are on different axes, intersecting cells that do not fall in the scope are still drawn. Their value is determined by the missingValueString GridBlox property. If no scope is defined, the calculated member is displayed wherever the members involved in the *expression* appear. For Microsoft Analysis Services, Alphablox Cube Server, and SAP BW data sources, use unique names as follows: • The dimension name needs to be enclosed in brackets and the cube name (MSAS, Alphablox Cube) or query name (SAP BW) needs to be included. • The member name needs to enclosed in brackets, or use the member's fully-qualified unique name, starting from the cube (MSAS, Alphablox Cube Server) or query (SAP BW). For example, – [California] – [My Cube].[Market].[West].[California] If the member name is not enclosed in brackets, it will be treated as a member whose name is all uppercase. The following member search functions are available for specifying the level of members the calculated member should be displayed for: • `Leaf()`: the leaf-level descendants of the specified member. Example: `Market: leaf(East)` (Essbase) • `Child()`: the children of the specified member. Example: `Market: child(East)` (Essbase) • `Descendants()`: all descendants of the specified member. Only one member can be specified in the function. Example: `[My Cube].[Market]:descendants([East])"` (MSAS, Alphablox Cube, SAP BW) • `Gen()`: all members of the specified generation. Example: `Market: gen(2)` • `Not()`: members to which the calculation should not be applied. Example: `[My Cube].[Market]: not([East], [West])` (MSAS, Alphablox Cube Server, SAP BW) There is another `Find()` function for finding members that meet the specified criteria. For example, `Find(Sales < 10000)`. The function names are case-insensitive. See the usage discussion of "Scoping" on page 173 and "Example 3: Adding a generation number and scope" on page 175.

Functions for CalculatedMembers: You can use functions in your calculation expression. Each function, except `Abs`, `Sqrt`, `Round`, `ifNotNumber`, `Power`, `Rank`, `RunningTotal`, and searching related functions, has the following syntax:

- *functionName(gen(generation))*. Calculate the value based on all the item members that are at a specified generation. Generation 1 means calculating the value based on root members. The example below creates a calculated member called "Standard Deviation" on the Product dimension with its value being the standard deviation of all members at generation 2.

  ```
  Product: Standard Deviation = Stdev(gen(2))
  ```

  Generation 0 means all generations are included in the calculation.
- *functionName(member1, member2, ..., memberN)*. Calculate the value based on the specified members in the dimension where the calculated member is added to. The example below adds a calculated member called "Standard Deviation" on the Product dimension with its value being the standard deviation of CD, Cassette, and TV

  ```
  Product: Standard Deviation = Stdev(CD, Cassette, TV).
  ```
- *functionName(searchFunction(member)).* Calculate the value based on the results from the search functions (`Child`, `Descendants`, `Leaf`, and `find`). The example below adds a calculated member called "Standard Deviation" on the Product dimension with its value being the standard deviation of all children of Audio.

  ```
  Product: Standard Deviation = Stdev(Child(Audio))
  ```
- A function can also take the result from another calculation. The example below adds a calculated member called "Absolute Total Values" on the Product dimension with its value being the absolute value of the total for generation 2 members.

  ```
  Product: Absolute Total Values = Abs(Sum(gen(2)))
  ```

The functions supported are as follows:

- "Arithmetic Functions" on page 168
- "Search Functions" on page 169
- "Special Calculation Functions" on page 170
- "Conditional and Missing Value Related Function" on page 172

Arithmetic Functions:

Arithmetic Functions	Description
`Abs`	Returns the absolute value of a number. This can only be used on a single number item such as the result of another calculation or a single member. For example: `Product: Average for Audio = Abs(Average(Audio)` `Product: Absolute Value for CD Sales = Abs(CD)`
`Average`	Returns the average of all the numbers in the definition, which is the `sum` divided by `count`. If the `count` is zero, the average is returned as a missing value.
`Count`	Returns the count of all numbers in the definition. Missing values are ignored. If there are no values to count, zero is returned.
`Max`	Returns the highest value in all the numbers in the definition.

`Median`	Returns the value of the number in the middle of the set; that is, half the numbers have values that are greater than the median, and half have values that are less.
`Min`	Returns the lowest value in all the numbers in the definition.
`Power`	Calculates the first parameter raised to the power of the second parameter. The parameters can either be member names or numeric constants.
`Product`	Returns the multiplication of all the values in the definition.
`Round`	Returns the integer part of the number rounded to the nearest whole number. This can only be used on a single number item such as the result of another calculation or a single member. For example: `Product: Total Sales = Round(Sum(gen(2)))` `Product: Rounding value for CD Sales = Round(CD)`
`Sqrt`	Returns the square root of a number. This can only be used on a single number item such as the result of another calculation or a single member. For example: `Product: My Calculation 2 = Sqrt(Average(gen(2)))` `Product: My Calculatioin 1 = Sqrt(CD)`
`Stdev`	Returns the standard deviation of all the numbers in the definition. The standard deviation is a measure of how widely values are dispersed from the average value (the mean).
`Sum`	Returns the addition of all the numbers in the definition. Missing values are ignored. If there are no values to add, zero will be returned.
`Var`	Returns the variance, which the average squared deviation of each number in the set from the average.

Note: For Microsoft Analysis Services and SAP BW data sources, specify the unique names when using the search functions.

Search Functions:

Search Functions	**Description**
`Child`	Returns all children of the specified member. For example: `Product: Average = Average(Child(Visual))` will calculate the average of all children of Visual. For Microsoft Analysis Services, Alphablox Cube, and SAP BW data sources, you must use the full path to the member names. For example, use `child([Time].[Calendar].[All Time Periods].[2000])` rather than `child([2000])`.

`Child`	Returns all descendants of the specified member. For example: `Product: Average = Average(Descendants(Visual))` will calculate the average of all descendants of Visual. For Microsoft Analysis Services, Alphablox Cube, and SAP BW data sources, you must use the full path to the member names. For example, use `descendants([Time].[Calendar].[All Time Periods].[2000])` rather than `descendants([2000])`.
`Leaf`	Returns all leaf-level descendants of the specified member. For example: `Product: Average = Average(Leaf(Visual))` will calculate the average of all leaf members of Visual. For Microsoft Analysis Services, Alphablox Cube, and SAP BW data sources, you must use the full path to the member names. For example, use `leaf([Time].[Calendar].[All Time Periods].[2000])` rather than `leaf([2000])`.
`Find`	Returns all members that meet the search criteria. For example: `Sum(Find(All Products:Rank>5)` calculates the sum of all whose ranking is lower than 5. Valid comparisons are >, <, >=, <=, =, and `!=` (<> can also be used for not equal tests). For Microsoft Analysis Services, Alphablox Cube, and SAP BW data sources, you must use the full path to the member names.

Special Calculation Functions: There are three special calculation functions: `LookupCount`, `Rank`, and `RunningTotal`.

Special Calculation Functions	Description
`LookupCount`	Similar to Excel's LOOKUP function, this function accepts two parameters (two strings) containing lists of comma separated numbers. The function looks in the first list for the specified value and returns a value from the same position in the second list. This function can be used to create custom statistical functions based on fixed value lookups. For example: D3() could be defined as `LookupCount("2,3,4,5", "8.3, 6.4, 4.3, 2.1")` If there is a count of 2 items with values in the D3() column, then the result would be 8.3. If there

are 5 items in the count, then the result of the calculation would be 2.1. If there are more than 5 or less than two, then the result would be NaN (Not a Number).

This function can be used to implement statistical functions required in calculated members to create control charts. Control charts are usually employed to visually look for variations to figure out whether a process is in control or out of control.

`Rank`

Returns the values from the specified dimensions in ascending or descending order for the specified member. The syntax for `Rank` is:

Rank(*member*, *dimension*, *generation*, *order*, *grouping*, *number*)

where:

- *member* is the member in this dimension which you want to rank
- *dimension* is the dimension on the opposite axis whose members will be used to generate the rank
- *generation* is the generation of the members of the dimension to be ranked. A generation of 0 means that all members are ranked.
- *order* is either `ASC` for ascending order or `DESC` for descending order. In descending order, the largest number will be ranked 1. In ascending order, the smallest number will be ranked 1.
- *grouping* is optional. If it is present and set to `GROUPDIM`, when there are multiple dimensions on the opposite axis, separate ranking will be done for each grouping of the dimension that is not part of the ranking definition. If it is not set or set to `NOGROUP`, ranking will be performed across groups. This will only have an effect when there is more than one dimension on the axis.
- *number* is optional. Specifies the number of items to rank. For example, specifying a number of 5 means to rank the top 5 members. Note that when this parameter is specified, the grouping parameter (`GROUPDIM` or `NOGROUP`) has to be specified as well.

See "Example 6: Adding ranking" on page 177 and "Example 7: Adding a separate ranking within each group" on page 177.

`RunningTotal`

Returns the accumulative sum of values from the specified dimension for the specified member. The syntax for `RunningTotal` is:

RunningTotal(*member*, *dimension*, *generation*, *grouping*)

where:

- *member* is the member in this dimension which you want to calculate the running totals for
- *dimension* is the dimension on the opposite axis whose members will be used to calculate the running totals
- *generation* is the generation of the members of the dimension to be calculated. A generation of 1 means to only include the root members in the calculation. A generation of 0 means that all members are to be included.
- *grouping* is optional, and if it is present and set to `GROUPDIM`, when there are multiple dimensions on the opposite axis, separate running totals will be done for each grouping of the dimension that is not part of the running total definition. This will only have an effect when there is more than one dimension on the axis.

See "Example 8: Adding running totals within each group" on page 178.

Conditional and Missing Value Related Function:

Conditional and Missing Value Related Functions

`If`

This function takes three parameters, separated by commas:

if(*condition*, *result_if_true*, *result_if_false*)

where *condition* has a left part and a right part, separated by one of the following operators: `<=`, `>=`, `=`, `<`, `>`, or `!=` (`<>` can also be used for not equal tests). For example:

```
Scenario:Act/Bdgt{MissingisZero} =
    If(Budget=0, 0, Actual - Budget*100 / Budget)
```

This will create a calculated member named "Act/Bdgt" in the Scenario dimension. The value for the calculated member will be zero if Budget is zero (or if Budget is missing as the `MissingIsZero` keyword indicates). If Budget is not zero, the value for "Act/Bdgt" will be Actual-Budget*100/Budget.

`ifNotNumber`

By default, missing or null values are treated as missing. You can substitute the function `ifNotNumber` for a member value to provide special case logic to handle missing or null values in the result set used in the calculation. The `ifNotNumber` function has the following syntax:

ifNotNumber(*memberName*,*value*)

where:

- *memberName* is the name of the member in which the function operates on.

- *value* is the numeric value which replaces the missing or null member value. The value specified must contain no commas.

Note: This function works on one member at a time. If you want missing values to be treated as zero for all members involved in calculations, use the keyword `missingIsZero`. See the usage discussion on "Calculation Involving Missing Values" on page 174. For an example, see "Example 4: Replacing missing or null values with the value 0" on page 175.

Usage

The following restrictions apply to calculated members:

- You must place double quotes around member names that contain special characters such as commas.
- You cannot specify relative positions for calculated members in Expand/Collapse mode.
- The values used in the calculation need to be in the result set in order for the added calculated member to display. For example, if a calculation involves generation 3 members, the calculated member will not display unless generation 3 members are in the result set.
- With Microsoft Analysis Services, SAP BW, and DB2 Alphablox cubes, the syntax must use unique member names.
- With relational data sources, the available "dimensions" are Record # and Columns.
- You cannot use the Keep/Remove option on calculated members, but you can use the Show/Hide option.

A unique name (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display name can be used for the dimension and member names specified in the property's value. This allows you to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, you can specify a member, regardless of the alias table in use, by using the base name.

To clear a calculated member, pass an empty string to the `setCalculatedMembers()` method.

Scoping: If you are creating multiple calculated members and want the scoping to either include or exclude other calculated members, the ordering of the calculated members is important and will affect the scope of the displayed results. For example, if you provide a scope to limit a calculated member to calculate only on a specific set of members, any calculated members that are created after the scoped calculated member (that is, calculated members that appear to the right in the *definitionString*) will be included in the initial scoped calculated member. This is because at the time the scoped calculated member is evaluated, the other calculated members do not exist and are therefore not removed from the scope. In these cases, you might want to put the definition for the non-scoped members used in other calculated members before the scoped calculated member. The following two definitions are not equivalent and would produce the results shown:

```
calculatedMembers="All Products: Product1 and Product2=Product1 + Product2
{Measures:Sales,COGS}, Measures:Gross Margin=Sales - COGS"
```

produces the following output:

	Sales	COGS	Gross Margin
Product1	500.00	300.00	200.00
Product2	1500.00	1000.00	500.00
Product1 + Product2	2000.00	1300.00	700.00

whereas:

```
calculatedMembers="Measures:Gross Margin=Sales - COGS, All Products:
Product1 and Product2=Product1 + Product2 {Measures:Sales,COGS}"
```

produces the following:

	Sales	COGS	Gross Margin
Product1	500.00	300.00	200.00
Product2	1500.00	1000.00	500.00
Product1 + Product2	2000.00	1300.00	

The difference between these two examples is in the first example, the scope of the calculated member `Product1 + Product2` will include the calculated member `Gross Margin` (because at the time of scoping, it did not exist); in the second example, the scope of the calculated member `Product1 + Product2` will not include the calculated member `Gross Margin`.

You can also use the member search functions to specify for whether the calculated member should be displayed for all children or all leaf-level descendants of a member or for members of a specific generation. The following example creates a calculated member Difference (the difference between Actual and Budget) on the Scenario dimension for all children of Products.

```
Scenario:Difference  = Actual - Budget {Products: Child(Products)}
```

Calculation Involving Missing Values: When the calculation involves missing values, by default they are treated as missing data and the calculation will fail. When missing data is displayed in a GridBlox, the grid cell will be blank by default. This is because GridBlox has a property called missingValueString, whose default value is an empty string. To treat all missing values as zero in your calculation, use the `missingIsZero` keyword. For example,

```
All Locations:East+Central {West:missingIsZero} = East + Central
```

adds a calculated member called “East+Central” before the member West, and all missing values are treated as 0 in the calculation. This keyword, however, only applies to member variables and has no effects on calculation functions.

If you want to treat missing values as missing for some of the members but not all, use the `ifNotNumber` function for each member that you want missing values to be treated as zero. For example,

```
All Locations:East+Central {West} = ifNotNumber(East,0) + Central
```

adds a calculated member called “East+Central” before the member West, and missing values in East will be treated as 0 while missing values in Central will be treated as missing. As a result, when Central contains missing values, the

calculation will fail and return missing, and an empty string will be displayed in those grid cells unless the GridBlox `missingValueString` property is set otherwise.

Note: If a calculation results in missing values in the entire row or column, then the row or column will not appear at all.

See "Example 4: Replacing missing or null values with the value 0" on page 175.

Examples

Example 1: Adding a calculated member named Profit at the end of the Measures dimension: The value in each cell of the Profit member is derived by subtracting the values in the corresponding cells of the Expenses and Sales members.

```
setCalculatedMembers("Measures : Profit = Sales - Expenses");
```

	Actual		**Budget**	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Expenses	*value*	*value*	*value*	*value*
Profit	*calc value*	*calc value*	*calc value*	*calc value*

Example 2: Specifying the position of the calculated member: You can position the calculated member Profit before the member Expenses in the grid by adding Expenses as a reference member:

```
setCalculatedMembers("Measures : Profit {Expenses} = Sales - Expenses");
```

	Actual		**Budget**	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Profit	*calc value*	*calc value*	*calc value*	*calc value*
Expenses	*value*	*value*	*value*	*value*

Example 3: Adding a generation number and scope:

```
setCalculatedMembers("Measures : Profit {Expenses:2} = Sales - Expenses
{Actual:West}");
```

	Actual		**Budget**	
	East	West	East	West
Sales	*value*	*value*	*value*	*value*
Profit	*#missing*	*calc value*	*#missing*	*#missing*
Expenses	*value*	*value*	*value*	*value*

Example 4: Replacing missing or null values with the value 0: Assume the following JSP tag:

```
calculatedMembers = "All Locations:East+Central {West:2} = East + Central"
```

This code produces a calculated member called "East+Central" that will be positioned before the member West, with the same level of indentation as generation 2 members. The output is as follows:

All Time Periods	All Locations	Truffles	Chocolate Blocks	Chocolate Nuts
2000	**Central**	6,119	29,068	
	East	883	3,679	
	East+Central	7,002	32,747	
	West	44,029	268,398	
	All Locations	51,031	301,145	

Since there is no data for Chocolate Nuts, there is also no data for the calculated member "East+Central" for Chocolate Nuts.

If we add the `missingIsZero` keyword to treat missing values as zero:

```
calculatedMembers = "All Locations:East+Central {West:3:missingIsZero} =
East + Central"
```

The output generated is as follows:

All Time Periods	All Locations	Truffles	Chocolate Blocks	Chocolate Nuts
2000	**Central**	6,119	29,068	
	East	883	3,679	
	East+Central	7,002	32,747	0
	West	44,029	268,398	
	All Locations	51,031	301,145	

Example 5: Calculations involving members from different dimensions: In the following JSP tag example, a calculated member "Percent Total" is added to the All Products dimension with values calculated by dividing the value of All Products by the intersection of All Products and All Locations:

```
calculatedMembers="All Products: Percent Total=
All Products / All Locations:All Locations; All Products:All Products"
```

The generated output is as follows:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Percent Total
2000	**Central**	72455.09	6849.592	107642.24	0.105
	East	10381.12	1620.36	14943.32	0.015
	West	593387.76	47641	905814.55	0.881
	All Locations	676223.97	56110.95	1028400.11	1
2001	**Central**	855542.96	29691.17	2384096.835	0.183
	East	6288893.64	15333.12	177250.755	0.136
	West	3266694.67	97033.65	8887288.195	0.681
	All Locations	4751131.27	142057.94	13043886.785	1
All Time Periods	**Central**	927998.05	36540.76	2491739.075	0.177
	East	639274.76	16953.48	1787445.075	0.127
	West	3860082.43	144674.65	9793102.745	0.695
	All Locations	5427355.24	198168.89	14072286.895	1

Each value in the Percent Total column is calculated using the values from All Products as the dividends and the values at the intersection of All Products and All Locations (1028400.11 for Year 2000 and 13043886.785 for year 2001) as the dividers.

Example 6: Adding ranking: In this example, a calculated member "Rank" is added to the All Products dimension, with the largest number for generation 2 members ranked first:

```
All Products:Rank = Rank(All Products, All Locations, 2, DESC)
```

The output generated is as follows:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
2000	Central	72455.09	6849.592	107642.24	8
	East	10381.12	1620.36	14943.32	9
	West	593387.76	47641	905814.55	7
	All Locations	676223.97	56110.95	1028400.11	
2001	Central	855542.96	29691.17	2384096.835	4
	East	6288893.64	15333.12	177250.755	6
	West	3266694.67	97033.65	8887288.195	2
	All Locations	4751131.27	142057.94	13043886.785	
All Time Periods	Central	927998.05	36540.76	2491739.075	3
	East	639274.76	16953.48	1787445.075	5
	West	3860082.43	144674.65	9793102.745	1
	All Locations	5427355.24	198168.89	14072286.895	

To rank only the top N or bottom N members, add an integer as the sixth parameter at the end. Note that in order to specify the number of ranking, the fifth parameter (NOGROUP or GROUPDIM) needs to be specified:

```
calculatedMembers="All Products:Rank = Rank(All Products, All Locations, 2,
DESC, NOGROUP, 5)"
```

Example 7: Adding a separate ranking within each group:

```
calculatedMembers = "All Products:Rank =
Rank(All Products, All Locations, 2, DESC, GROUPDIM)"
```

The above example generates the following output:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
2000	Central	72455.09	6849.592	107642.24	2
	East	10381.12	1620.36	14943.32	3
	West	593387.76	47641	905814.55	1
	All Locations	676223.97	56110.95	1028400.11	
2001	Central	855542.96	29691.17	2384096.835	2
	East	6288893.64	15333.12	177250.755	3
	West	3266694.67	97033.65	8887288.195	1
	All Locations	4751131.27	142057.94	13043886.785	

All Time Periods	All Locations	Specialties	Seasonal	All Products	Rank
All Time Periods	Central	927998.05	36540.76	2491739.075	2
	East	639274.76	16953.48	1787445.075	3
	West	3860082.43	144674.65	9793102.745	1
	All Locations	5427355.24	198168.89	14072286.895	

A calculated member "Rank" is added to the All Products dimension based on the values of generation 2 members in the All Locations dimension, with the largest number ranked first and separate ranking within each group in the dimension.

Example 8: Adding running totals within each group:

```
calculatedMembers = "All Products:Running Totals =
RunningTotal(All Products, All Locations, 2, GROUPDIM)"
```

The above example generates the following output:

All Time Periods	All Locations	Specialties	Seasonal	All Products	Running Totals
2000	Central	72455	6850	107642	107642
	East	10381	1620	14943	122586
	West	593388	47641	905815	1028400
	All Locations	676224	56111	1028400	
2001	Central	855543	29691	2384097	2384097
	East	6288894	15333	1772502	4156599
	West	3266695	97034	8887288	13043887
	All Locations	4751131	142058	13043887	
All Time Periods	Central	927998	36541	2491739	2491739
	East	639275	16953.48	1787445	4279184
	West	3860082	144675	9793102.745	14072287
	All Locations	5427355	198169	14072287	

A calculated member called "Running Totals" is added to the All Products dimension, which contains the accumulative sum for each group of the generation 2 members on the All Locations dimension.

See Also

"Inputting and Modifying Data" in the *Developer's Guide*.

catalog

Specifies the database catalog for this DataBlox.

Data Sources

All

Syntax

```
catalog=catalog
```

where:

Argument	Default	Description
`catalog`	empty string	Name of a database catalog.

Usage

The value for `catalog` is one of the values provided when defining a data source to Analysis Server. If no `catalog` property is specified on a DataBlox, the value is taken from the corresponding data source definition, if the value is present.

A catalog is known as an "application" in IBM DB2 OLAP Server or Hyperion Essbase terminology.

columnSort

Specifies how to sort data values for members on the column axis.

Data Sources

All

Syntax

`columnSort=`*`sortString`*

where:

Argument	Default	Description
`column`	none	A column in a relational result set.
`ascending`	none	Specify `true` to sort ascending; `false` to sort descending.
`tuple`	none	The tuple on the column axis that specifies the column to be sorted.
`dimension`	none	The dimension on the row axis for which grouping is preserved. Specify `null` to indicate no grouping is to be preserved on the row axis.
`preserveHierarchy`	`false`	Specify `true` to preserve the hierarchy in the row axis, keeping members with their parents after the sort operation; `false` to not preserve hierarchy.

Argument	Default	Description
sortString	none	A comma-delimited string in one of the following formats: • *tupleIndex*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction*, *preserveHierarchy* *tupleIndex* —string representation of an integer, representing the zero-based tuple index member (column) to sort, where `0` indicates the leftmost column. *groupingNestLevel* —string representation of an integer, representing the dimension on the row axis for which grouping is preserved. For example, if Product and Market are on the row axis, a value of 1 sorts into sequence within the Market dimension. Specify `-1` to sort without regard to row groupings. The default is `-1`. *direction* —a case-insensitive string of either `"Ascending"`, `"Asc"`, `"Descending"` or `"Desc"`. *preserveHierarchy* —string representation of a boolean. see the `preserveHierarchy` argument. Defaults to `false`. For example: `setColumnSort("1,0,asc,true");` `setColumnSort("1,0,asc");` `setColumnSort("0,descending");`

Usage

The `getColumnSort` method returns a string of four comma-separated items: *tupleIndex*, *groupingNestLevel*, *direction*, and *preserveHierarchy*.

The following screen shots show the results of an ascending sort operation on the Qtr1 column depending on 1) whether the hierarchy is preserved , and 2) whether the grouping within the a specified level/dimension is preserved. The first example preserves the hierarchy in the Market dimension, yet not the grouping within the Product dimension.

Product	Market	Qtr1
200	East	562
	New Mexico	-14
	Louisiana	336
	Oklahoma	468
	Texas	675
	South	1465
	West	2325
	Central	2369
	Market	6721
100	West	1042
	New Mexico	-27
	Oklahoma	171
	Louisiana	212
	Texas	695
	South	1051
	Central	2208
	East	2747
	Market	7048

The following example preserves neither the hierarchy in the Market dimension nor the grouping in the Product dimension.

Product	Market	Qtr1
100	New Mexico	-27
200	New Mexico	-14
100	Oklahoma	171
	Louisiana	212
200	Louisiana	336
	Oklahoma	468
	East	562
	Texas	675
100	Texas	695
	West	1042
	South	1051
200	South	1465
100	Central	2208
200	West	2325
	Central	2369
100	East	2747
200	Market	6721
100	Market	7048

Examples

The following example demonstrates the use of the `columnSort` tag attribute:

```
columnSort="1, 0, asc"
```

See Also

"rowSort" on page 200, DataBlox `removeColumnSort()` method in the Javadoc documentation

connectOnStartup

Specifies whether the DataBlox will automatically connect to its data source upon Blox instantiation.

Data Sources

All

Syntax

`connectOnStartup="`*`connectOnStartup`*`"`

where:

Argument	Default	Description
`connectOnStartup`	`true`	A boolean argument. Specify `true` to automatically connect to the data source when the Blox is instantiated; `false` to not connect.

Usage

To prevent a DataBlox from connecting to a data source, set this property to `false`. When the query property is set later, you need to call the `connect()` method to connect to the data source.

When `connectOnStartup` is set to `true`, it overrides the `autoConnect` property. The `connectOnStartup` property causes a database connection, even if no query is defined.

See Also

"query" on page 199, "autoConnect" on page 162, "autoDisconnect" on page 163

credential

Specifies a credential object to be used with the given data source instead of a username and password

Data Sources

DB2 OLAP Server and Hyperion Essbase

Syntax

```
credential="credential"
```

where:

Argument	Default	Description
credential		The credential object to send to the data source.

Usage

This property accepts a credential object that DB2 Alphablox will pass to a DB2 OLAP Server or Hyperion Essbase data source instead of a username and password. You can utilize the single sign-on feature of DB2 OLAP Server or Hyperion Essbase to allow a user to log into the data source without storing the username and password in the DB2 Alphablox repository. Through Hyperion's Common Security Services, you can retrieve a token to pass to DB2 Alphablox. When the `credential` attribute is set, username and password settings in DB2 Alphablox for that data connection will be ignored. This property will not be saved with any bookmark.

Examples

The following example snippet shows how a token is retrieved from Common Security Services and passed to a DataBlox.

```
<%@ page import="com.hyperion.css.CSSAPIIF"%>
<%@ page import="com.hyperion.css.CSSException"%>
<%@ page import="com.hyperion.css.CSSSystem"%>
<%@ page import="com.hyperion.css.application.CSSApplicationIF"%>
<%@ page import="com.hyperion.css.common.CSSUserIF"%>
<%@ page import="java.io.*"%>
<%@ page import="java.net.*"%>
<%@ page import="java.util.*"%>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ page contentType="text/html;charset=utf-8" %>

<%!
public class MyCssApp implements CSSApplicationIF {
   //implements your application contract. Code omittend here.
}
%>

<html>
<head>
<blox:header/>
</head>

<%
    String credential = request.getSession().getAttribute("SSOToken");

   if (credential == null)
```

```
   {
      HashMap css_context = new HashMap();
      String user = request.getParameter("username");
      String password = request.getParameter("password");

      MyCssApp myApp = new MyCssApp();
      CSSSystem system = CSSSystem.getInstance();
      CSSAPIIF css = system.getCSSAPI();

      css_context.put(CSSAPIIF.LOGIN_NAME, user);
      css_context.put(CSSAPIIF.PASSWORD, password);
      css.initialize(css_context, myapp);

      CSSUserIF css_user = css.authenticate(css_context);

      credential = css_user.getToken();
      request.getSession().setAttribute("SSOToken", user1.getToken());
   }
%>

<blox:data id="myData" credential="<%=credential %>"
    dataSourceName="EssbaseSSO" useAliases="true"
    query="!" visible="false" />

<body>
<blox:present id="myPresent" width="700" height="500">
   <blox:data bloxRef="myData" />
</blox:present>
</body>
</html>
```

dataSourceName

Identifies the external data source that this DataBlox accesses.

Data Sources

All

Syntax

`dataSourceName="`*`dataSourceName`*`"`

where:

Argument	Default	Description
`dataSourceName`	empty string	A data source name as defined in DB2 Alphablox.

Usage

The data source name must be defined in DB2 Alphablox. If you do not specify a data source, the Blox loads with no initial data source. This feature enables the data source to be set programmatically, perhaps based on user properties or actions. However, to prevent an error message appearing to the user, be sure to set the value of the autoConnect property to `false`.)

This `setDataSourceName()` method also reads in the properties of the data source such as username, password, catalog, schema, query, and dimensions on the page axis. Therefore, if you want to set any of these properties using the Java methods such as `setUserName()` and `setPassword()`, set them after the `setDataSourceName()` method.

Tip: The order these data source properties are set is not an issue if you use the Blox tags. Because the tags are designed to enforce that the data source is set before the call to set the other data source properties, the side effect is taken care for you.

Tip: For security reasons, it is not a good idea to provide values for the `userName` and `password` properties on a JSP page. For more information on these properties, see "userName" on page 208 and "password" on page 197. For related information, see also "query" on page 199, "schema" on page 201, "useAASUserAuthorizationEnabled" on page 207, "provider" on page 198, and "catalog" on page 178.

Examples

The following example shows the custom tag attributes with values for an IBM DB2 OLAP Server or Hyperion Essbase data source.

```
dataSourceName="MyEssbaseDataSource"
schema="Basic"
catalog="Demo"
query="<SYM <ROW (Product) <ICHILD Product <COL (Market) <ICHILD Market !"
```

The following example shows the properties with values for an Alphablox cube.

```
dataSourceName="MyAlphabloxCube"
query="SELECT Measures.MEMBERS ON COLUMNS,
         {[Store].[Store State].[CA], [Store].[Store State].[WA]}
         ON ROWS
       FROM [Sales]"
```

The following example shows the properties with values for a Microsoft OLAP Services data source.

```
dataSourceName="MyOLAPDataSource"
catalog="MySchema"
schema="MyCatalog"
query="SELECT Measures.MEMBERS ON COLUMNS,
         {[Store].[Store State].[CA], [Store].[Store State].[WA]}
         ON ROWS
       FROM [Sales]"
```

The following example shows the properties with values for an IBM DB2 UDB data source.

```
dataSourceName="MyDB2"
catalog="MyCatalog"
query="SELECT Actual.SalesQty, Actual.ProductID FROM MyCatalog"
```

The following example shows the properties with values for an Oracle data source.

```
dataSourceName="MyOracleDataSource"
schema="MySchema"
catalog="MyCatalog"
query="SELECT Actual.SalesQty, Actual.ProductID,
          Projected.SalesQty, Projected.ProductID
       FROM Actual, Projected
       WHERE Actual.SalesQty <Projected.SalesQty"
```

The value for the `dataSourceName` string must be a data source already defined to DB2 Alphablox. For information on setting up data sources, see the *Administrator's Guide*.

See Also

"catalog" on page 178, "query" on page 199, "schema" on page 201, "useAASUserAuthorizationEnabled" on page 207

dimensionRoot

Specifies the dimension and a single member to use as the root.

Data Sources

Multidimensional

Syntax

```
dimensionRoot="dimensionNameAndNewRootMember"
```

where:

Argument	Default	Description
dimensionNameAndNewRootMember	empty string	A String which specifies the new root member for the specified dimension or dimensions. The String is in the form: `"DimA:NewRootMemberA;DimB:NewRootMemberB;"` If you specify a dimension without a new root member, the dimension root is reset to the database default root member for that dimension.
dimension	none	A Dimension object. If `dimension` appears in a page filter, in the Member Filter, or if a member is `null`, then the database uses its default dimension root. acts as the root of the dimension.
member	none	A Member object. If `member` is `null`, the dimension root resets to the database default.

Usage

If the named dimension appears in a page filter or the Member Filter, the selected member acts as the root of the dimension. If there is conflict between this property value and the query, the query overrides the property.

A unique name (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display name can be used for the dimension and member name string specified in the property's value. This allows assemblers to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, an assembler can specify a member, regardless of the alias table in use, by using the base name.

Multiple dimensions can be specified, but only one member per dimension.

The `String getDimensionRoot()` method returns a String of the form:

```
"DimA:RootMemberA;DimB:RootMemberB;"
```

Examples

The following example specifies, as an attribute to the DataBlox custom tag, that the root of the `Products` dimension is `Tools` and the root of the `Market` dimension is `East`.

```
dimensionRoot="Products: Tools;Markets: East"
```

See Also

"query" on page 199

drillDownOption

Specifies the type of drill operation to perform.

Data Sources

Multidimensional

Syntax

```
drillDownOption="drillDown"
```

where:

Argument	Default	Description
drillDown	1	An integer from 1 to 5 specifying the level to drill down to. Possible values are: 1: Drill down to next generation 2: Drill down to all descendants (same as "Expand All") 3: Drill down to bottom generation 4: Drill to siblings 5: Drill to same generation

Usage

For explanations of the terms such as descendants, siblings, and generation, see "OLAP Terms and Concepts" in the *Administrator's Guide*.

Examples

```
drillDownOption="4"
```

See Also

"drillKeepSelectedMember" on page 186, "drillRemoveUnselectedMembers" on page 187, DataBlox `drillDown()` and `drillToAllDescendants()` methods

drillKeepSelectedMember

Specifies whether the member being drilled on should be retained or removed when the display repaints.

Data Sources

Multidimensional

Syntax

```
drillKeepSelectedMember="keepSelected"
```

where:

Argument	Default	Description
keepSelected	true	A boolean argument. Specify `true` to retain the member being drilled on; `false` to remove it.

Examples

```
drillKeepSelectedMember="false"
```

See Also

"drillRemoveUnselectedMembers" on page 187

drillRemoveUnselectedMembers

Specifies whether to remove all members that are not being drilled on when the display repaints.

Data Sources

Multidimensional

Syntax

```
drillRemoveUnselectedMembers="removeUnselected"
```

where:

Argument	Default	Description
removeUnselected	false	Specify true to remove all members that are not being drilled on, false to keep them.

Examples

```
setDrillRemoveUnselectedMembers(true);
```

See Also

"drillKeepSelectedMember" on page 186

enableKeepRemove

Specifies whether the Keep Only and Remove Only options are available to the end user in the context menus of both the GridBlox and ChartBlox.

Data Sources

All

Syntax

```
enableKeepRemove="enable"
```

where:

Argument	Default	Description
enable	false	Specify true to enable the Keep Only and Remove Only options; false to disable them.

Usage

The Keep Only and Remove Only options give end users the ability to control which members and columns are visible in the grid.

Even if you have enabled or disabled the Remove Only and Keep Only option using this property, the end user can enable or disable it by using the "Enable keep and remove" checkbox under the Data tab of the Options dialog.

Examples

```
enableKeepRemove="true"
```

See Also

"enableShowHide" on page 188

enableShowHide

Specifies whether the Show Only, Show All, and Hide Only options are available to end users in the context menus of both the GridBlox and ChartBlox.

Data Sources

All

Syntax

```
enableShowHide="enable"
```

where:

Argument	Default	Description
enable	true	Specify true to enable the Show Only, Show All, and Hide Only options; false to disable them.

Usage

The Show/Hide options give you and the end user the ability to control which members are visible in the grid. It does not replace the Keep/Remove options, although its behavior is similar. You can use the Show/Hide feature in the same places as Keep/Remove.

The end user can turn the Show/Hide feature on and off through the Options dialog box under the Data tab. The options available when it is enabled are to Show or Hide members individually, and to Show All members of a dimension. The user cannot selectively show members once they are hidden, or hide all members. Each dimension must contain one member at all times.

Hidden members continue to appear in the Member Filter. Additionally, saving a bookmark when certain members are hidden preserves their hidden state.

Show/Hide vs. Keep/Remove

The Show/Hide feature differs from the Keep/Remove feature in several ways. The Show/Hide feature preserves the hidden state of members through subsequent GUI operations until they are unhidden through Show All. For example, the user can drill up and back down to the hidden member's level, and the member will stay hidden. This is in contrast to the Keep/Remove functionality, where under the same circumstances the removed member would be revealed.

Show/Hide does not interfere with the output of DB2 Alphablox calculated members. Although a member may be hidden from view, the data is still accessible for use in calculations. When a member is removed using Keep/Remove, the calculated member can no longer access the removed data and will return the value "#missing" (or whatever missing value string you have specified).

Examples

```
enableShowHide="true"
```

See Also

"enableKeepRemove" on page 187

hiddenMembers

Specifies which members to hide using the Show/Hide functionality.

Data Sources
All

Syntax
`hiddenMembers="membersToHide"`

where:

Argument	Default	Description
`membersToHide`	empty string	List of initially hidden members. Format the string as follows: `DimensionName1:MemberNameA,MemberNameB;` `DimensionName2:MemberNameD,MemberNameD;` `...` `DimensionNameN:MemberNameX,MemberNameY` The semicolon is required to separate members in different dimensions. **Note:** If you merge multiple dimension headers into one using "mergedHeaders" on page 194, since mergedHeaders are performed first, you should use the newly merged headers when specifying the members to hide.

Usage
One member of each dimension must remain in the grid. If you specify for all members of a dimension to be hidden, the last member specified will not be hidden. It is best to make sure that you do not remove all members of any dimension.

Examples
If you want to hide the members *East* and *North* on the *Market* dimension and *Audio* on the *Product* dimension, you would define the `hiddenMembers` property as follows:

```
hiddenMembers="Market:East,North; Product:Audio"
```

See Also
"enableShowHide" on page 188, DataBlox `hideMembers()` and `showMembers()` methods.

hiddenTuples
Specifies which tuples in the result set to hide using the Show/Hide functionality.

Data Sources
Multidimensional

Syntax
`hiddenTuples="selectedTuples"`

where:

Argument	Default	Description
`selectedTuples`	empty string	List of initially hidden tuples. Format the string as follows: `Dimension1,Dimension2,...,DimensionN:` `Dim1Member1, Dim2Member1,..,DimNMember1;` `Dim1Member2,Dim2Member2,...,DimNMember2;...` `Dim1MemberM,Dim2MemberM,...,DimNMemberM` Each tuple list contains the dimension names separated by commas, followed by a colon, followed by the list of tuples. Each tuple consists of one member from each of the dimensions specified and is separated by a semicolon. Each member of the tuple is separated by a comma. You can also specify multiple tuple lists, with each list enclosed in curly braces and separated by a comma. This allows you specify tuples from dimensions on the opposite axis in the same syntax. See the Usage section below for details. **Note:** If you merge multiple dimension headers into one using "mergedHeaders" on page 194, since mergedHeaders are performed first, you should use the newly merged headers when specifying the tuples to hide.

Usage

In order for a specified tuple to be hidden, the tuple must exist in the result set.

You can specify tuples from the opposite dimension in the same syntax by enclosing the tuple list from dimensions on one axis in curly braces and separate the list by a comma:

```
{Period,Product:Q1,Audio;Q2,Visual},{Accounts,Market:Profit,East}
```

The above example will generate an output as follows:

		Margin				Profit		
Period	Product	East	West	South	Market	West	South	Market
Q1	Visual	4950.9	23995	15344	44289.9	8042	4474	4373.9
	Product	1461	37890	15344	54495	12917	4474	834
Q2	Audio	9331	13354		22685	4320	0	6852
	Product	28232	36785	14895	79912	11449	4199	22924
Q3	Audio	9390	13745		23135	5324	0	8300
	Visual	22282	24740	15675	62697	8946	5430	22680
	Product	31672	38485	15675	85832	14270	5430	30980

Note: The `setHiddenTuples()` method sets the new list of hidden tuples, overriding any hidden tuple list set earlier. To hide additional tuples, use the DataBlox `hideTuples()` method.

See Also

DataBlox `hideTuples()` method

internalSortEnabled

Specifies whether to disable the Alphablox internal sorting mechanism while the sorting indicators in the Blox user interface are still available.

Data Sources

Multidimensional

Syntax

```
internalSortEnabled="enabled"
```

where:

Argument	Default	Description
enabled	true	Specify false to disable the Alphablox internal sorting mechanism but still show the sorting indicators in the Blox user interface.

Usage

This property is useful when you want to write your own sorting code for custom sorting behaviors but still use the sorting indicators in the user interface to track and control the sorting.

leafDrillDownAvailable

Specifies whether the user should be allowed to drill down on a leaf member.

Data Sources

Multidimensional

Syntax

```
leafDrillDownAvailable="available"
```

where:

Argument	Default	Description
available	false	Specify true to enable leaf drill downs; false to disable them.

Usage

This property is useful only when you want to perform custom actions when a user drills down on a leaf member. In this case only, you need to set the value to true. If you do not need to enable drilling down on a leaf member to call a function, leave this property at its default value of false.

memberNameRemovePrefix

Specifies the start point of a member name when returned from the data source.

Data Sources

Multidimensional

Syntax

```
memberNameRemovePrefix="prefix"
```

where:

Argument	Default	Description
prefix	empty string	Start point of the member name.

Usage

The memberNameRemovePrefix property removes the text from a member name string returned from a data source beginning with and including the specified string.

Note: This method only affects the result set, not the metadata. That is, subsequent metadata calls to get the display name of a member will still include the prefix.

This property is of particular use with IBM DB2 OLAP Server or Hyperion Essbase data sources where member names must be unique. Unique names are often created by adding unique strings as suffixes or prefixes on member names. Using this property enables stripping off the prefix strings before displaying the member names.

The removal can only be applied to member names; it cannot be applied to dimension names. Additionally, properties and methods that take member names as arguments will use the unique member name prior to the prefix or suffix removal.

Examples

If the memberNameRemovePrefix property is ″##″, the member ″123##Year″ will be displayed as ″Year″.

This property can be used with the memberNameRemoveSuffix property. For example, if the memberNameRemovePrefix string is ″$$″ and the memberNameRemoveSuffix string is ″##″, then the member ″123$$Year##978-9″ will be displayed as ″Year″.

See Also

“memberNameRemoveSuffix” on page 192

memberNameRemoveSuffix

Specifies the end point of a member name when returned from the data source.

Data Sources

Multidimensional

Syntax

memberNameRemoveSuffix="*suffix*"

where:

Argument	Default	Description
suffix	empty string	End point of the member name.

Usage
The memberNameRemoveSuffix property removes the text from a member name string returned from a data source beginning with and including the specified string.

Note: This method only affects the result set, not the metadata. That is, subsequent metadata calls to get the display name of a member will still include the suffix.

This property is of particular use with IBM DB2 OLAP Server or Hyperion Essbase data sources where member names must be unique. Unique names are often created by adding unique strings as suffixes or prefixes on member names. Using this property enables stripping off the suffix strings before displaying the member names.

The removal can only be applied to member names; it cannot be applied to dimension names. Additionally, properties and methods that take member names as arguments will use the unique member name prior to the prefix or suffix removal.

Examples
If the memberNameRemoveSuffix is ″##″, the member ″Year##978-9″ will be displayed as ″Year″.

This property can be used with the memberNameRemovePrefix property. For example, if the memberNameRemovePrefix string is ″$$″ and the memberNameRemoveSuffix string is ″##″, then the member ″123$$Year##978-9″ will be displayed as ″Year″.

See Also
“memberNameRemovePrefix” on page 191

mergedDimensions
Specifies whether multiple hierarchies of a dimension should be merged in the Other axis in the Data Layout panel and in Member Filter.

Data Sources
Microsoft Analysis Services, SAP BW

Syntax
```
mergedDimensions="dimensionString"
```

where:

Argument	Default	Description
dimensionString	null	A comma-delimited String representing the prefix of the dimensions to merge in the Other axis of the Data Layout panel

Usage
Microsoft Analysis Services and SAP BW support multiple hierarchies, allowing alternate views of cube data. Multiple hierarchies are two or more dimensions with names that share the same prefix followed by a period but have different suffixes. For example, Time.Calendar and Time.Fiscal are two different dimensions, but if

you merge them into a "logical" dimension (which does not actually exist), you can enhance the usability of your application as your users are less likely to be confused.

Once multiple hierarchies are merged, they appear as one dimension in the user interface in the Other axis in the Data Layout panel. For example, if you specify to merge all dimensions with the prefix "Time," Time.Calendar and Time.Fiscal will appear as a "Time" dimension in the Data Layout panel. When a user drags the Time dimension to the Row, Column, or Page axis, a dialog automatically pops up, asking the user to select one of the two hierarchies she wants to use. In Member Filter, all corresponding hierarchies are displayed under the Time dimension, but users can only select from one hierarchy.

Note: When you access the dimensions through methods such as the MDBMetaData object's `resolveDimension()` method, you should specify the actual dimension names ([Time].[Calendar] and [Time].[Fiscal], for example) that are actually stored in the data source. Since the merged dimension does not actually exist in the data source, using the merged dimension name will result in an error. To find out the dimensions that make up the merged dimension, use the `getCube().getMultipleHierarchies()` method.

Examples

The following example shows how to merge all hierarchies with the prefix "Time" into a non-existing dimension called Time, and all hierarchies with the prefix "Products" into a non-existing dimension called Products:

```
mergedDimensions="Time,Products"
```

See Also

`getCube().getMultipleHierarchies()` in the Javadoc documentation.

mergedHeaders

Specifies the dimensions on the same axis whose headers are to be merged.

Data Sources

All

Syntax

`mergedHeaders="`*`mergedString`*`"`

where:

Argument	Default	Description
`mergedString`	`null`	A colon-separated string of *dimensionString:matchPatterns:drillableDim* See the Usage section below for details.

Usage

- *dimensionString*—This string specifies the dimensions whose headers are to be merged and the new member name for the merged header, in the format of:

  dimensionList = newMemberName

  dimensionList is a comma-separated list of dimensions whose headers are to be merged. *newMemberName* is the name of the merged header. This is the name to use if you want to hide a member, row, or tuple (using DataBlox's

hiddenMembers or hiddenTuples property). By default, the merged header adds a space as a separator among the merged dimension headers. For example, if the headers for Scenario and Measures are merged, the new header is ″Scenario Measures,″ with a space in between.

Note: The order of the dimension specification has to be the same as that in the result set, and they must be consecutive. For example, if you have a query that returns All Time Periods, Measures, and Scenario in that sequence, then the following examples are valid:

```
mergedHeaders="All Time Periods, Measures, Scenario = Measures and
Scenario by Year" mergedHeaders="Measures, Scenario = Measures &
Scenario"
```

But the following are not:

```
mergedHeaders="All Time Periods, Scenario = Scenario by Period"
(dimensions are not consecutive) mergedHeaders="Measures, All Time
Periods = Measures by Period" (order is not correct)
```

Note: Calculated member (specified using the calculatedMembers property) is performed before the headers are merged. Therefore, when you need to add a calculated member or members, use the original dimension names. hiddenMembers and hiddenTuples, on the contrary, are performed after mergedHeaders, and therefore the newly merged header should be used.

- *matchPatterns*—Optional; A comma-separated list of pairs of the header pattern to match and the replacing header. Each pair of old header and new header should be in the format of *olderHeader = newHeader.* The following example merges the headers for Measures and All Time Periods, and replaces the string “Qtr 1” found in any header with the string “Q1,” “Qtr 2” with “Q2 ,” “Qtr 3” with “Q3,” and “Qtr 4” with “Q4.”

  ```
  mergedHeaders="All Time Periods, Measures = Measures by Period:  Qtr 1 =
  Q1, Qtr 2 = Q2, Qtr 3 = Q3, Qtr 4 = Q4"
  ```

  As a result, Qtr 1 00 Sales becomes Q1 00 Sales and Qtr 1 01 Forecast becomes Q1 01 Forecast.
- *drillableDim*—Optional; The dimension that is drillable when users drill on the merged header. If a drillable dimension is not specified, the first dimension listed in the *dimensionString* is by default the drillable dimension.

  Note: If a drillable dimension is specified without any match patterns, the colon separating the two strings should still be included. For example:

  ```
  mergedHeaders="Measures, Scenario::Scenario"
  ```

  Note: There can only be one drillable dimension when headers are merged. Setting a drillable dimension makes the other dimensions in the *dimensionString* not drillable.

Examples

The following example merges the headers for dimensions All Time Periods and Measures, with the new merged dimension name being Measures by Period. Four header name replacement match patterns are specified. The drillable dimension is set to All Time Periods (which is also the default if not specified, since it is the first dimension in the list).

```
mergedHeaders="All Time Periods, Measures= Measures by Period:
Qtr 1 = Q1, Qtr 2 = Q2, Qtr 3 = Q3, Qtr 4 = Q4:
All Time Periods"/>
```

onErrorClearResultSet

Specifies whether the existing result set should be cleared if a subsequent database operation fails.

Data Sources

All

Syntax

```
onErrorClearResultSet="clearResultSet"
```

where:

Argument	Default	Description
clearResultSet	false	Specify true to clear the result set; false not to clear it.

parentFirst

Specifies how the parents are returned relative to the children.

Data Sources

Multidimensional

Syntax

```
parentFirst="parentFirst"
```

where:

Argument	Default	Description
parentFirst	Respect the order of the members returned from the query	In Blox tags, specify true to place the parents before the children; false to place children first. If this tag attribute is not specified, the order of the members returned from the query is respected. The related Java methods take and return integers. setParentFirst() takes the following values: • DataBlox.PARENT_DEFAULT— the order of the members returned from the query should be respected • DataBlox.PARENT_FIRST— parent members should come before their child members regardless of the order of the members returned from the query • DataBlox.PARENT_LAST— parent members should come after their child members regardless of the order of the members returned from the query

Usage

In Blox tags, setting the value to true will return the data with the parent first (above or to the left of the children); setting the value to false will return the data with the parent last (below or to the right of the children). If this attribute is not specified in your DataBlox, the order of the members returned from the query is respected. Note that if you want to use the GridBlox's expand/collapse mode (expandCollapseMode ="true") and want parents to display first, set parentFirst to true rather than do so in the query. This is to ensure the expand/collapse mode can search through the result set correctly to determine the base members and shared members.

Important: If you set to have parent members come before or after the members returned, you cannot reset to respect the default order.

Examples

The following example demonstrates how to set the parent members to come before the children using JSP tags and Java method:

```
<blox:data ..
   parentFirst="true" />
<% myDataBlox.setParentFirst(DataBlox.PARENT_FIRST); %>
```

The following example demonstrates how to get the current order of parents and their children:

```
<% String message;
   int order;
   order = myDataBlox.getParentFirst();
   if (order == myDataBlox.PARENT_FIRST) {
      message = "Parent First";
   } else if (order == myDataBlox.PARENT_LAST) {
      message = "Parent Last";
   } else message="Default Order";
   out.write("The current parent-child order is: " + message);
%>
```

password

Specifies the database password to use when accessing the data source.

Data Sources

All

Syntax

`password="password"`

where:

Argument	Default	Description
`password`	empty string	Password for accessing the data source.

Usage

A default password is one of the values provided when defining a data source to DB2 Alphablox. If no `password` property is specified on a DataBlox, the value is taken from the data source definition, unless `AASUserAuthorizationEnabled` is set to `true` (in which case, the password the user entered is used).

If you use the associated `setPassword()` method in conjunction with `setDataSourceName()` method, you should set the password after calling `setDataSourceName()`. Otherwise, the DataBlox will connect with all the properties in the data source specified and override any properties set earlier. This is because the `setDataSourceName()` method also reads in the properties of the data source such as username, password, catalog, schema, query, and dimensions on page axis. Therefore, if you want to set any of these properties using the Java methods, set them after calling `setDataSourceName()`.

Tip: The order these data source properties are set is not an issue if you use the Blox tags. The tags are designed to enforce that the data source is set before the call to set the other data source properties, the side effect is taken care for you.

Examples

```
password="secret"
```

See Also

"dataSourceName" on page 183, "userName" on page 208

performInAllGroups

Specifies whether a drill operation is performed on all occurrences of the selected member in each outer nested group containing the dimension, or only on the single selected occurrence of the member.

Data Sources

Multidimensional

Syntax

```
performInAllGroups="perform"
```

where:

Argument	Default	Description
`perform`	`true`	Specify `true` to make a drill apply to all occurrences of the selected member; `false` to drill only the single occurrence of the member.

Usage

Even when this property is set to `true`, the member name must be the same in the other groups for the drill to occur. For example, assume Period is nested within Product. A drill on Qtr1 in VCR expands Qtr1 in TV because the member names are the same. However, Qtr2 through Qtr4 in VCR and TV are not expanded because the member names are different.

provider

Sets the provider string to use to access the data source.

Data Sources

OLE DB for OLAP only

Syntax

```
provider="provider"
```

where:

Argument	Default	Description
`provider`	String specified in data source definition	The name of the database supplier. For example, for Microsoft Analysis Server, the string is `MSOLAP`.

Usage

The default provider string of OLE DB for OLAP data sources is specified when the data source is defined to Alphablox. If the `provider` property is not specified for a DataBlox, the value is taken from the data source definition.

If you use the `setProvider()` Java method, you should set the provider string after calling `setDataSourceName()`. Otherwise, the DataBlox will connect with all the properties in the data source specified and override any properties set earlier. This is because the `setDataSourceName()` method also reads in the properties of the data source such as username, password, catalog, schema, query, and dimensions on the page axis. The order these data source properties are set is not an issue if you use the Blox tags because the tags are designed to enforce the proper order.

query

Specifies the initial query string that is passed to the data source.

Data Sources

All

Syntax

`query="queryString"`

where:

Argument	Default	Description
`queryString`	empty string	Query statement in the language understood by the data source. For relational data sources, use a SQL SELECT statement. For multidimensional data sources, use the appropriate language, such as Microsoft MDX or Essbase Report Specifications.

Usage

The `getQuery()` method returns the last query string that has been set for the current data source. User actions, such as sorts or drills, performed since the last query are not reflected in the returned value.

The `setQuery()` method sets the query string. The query is executed when the `connect()` method is called.

Examples

For an example of a query using the Microsoft MDX query language, see the MDX Statement in "Retrieving Data" of the *Developer's Guide*. For specific information from Microsoft on MDX,, see the following web links:

`http://www.microsoft.com/data/oledb/`

and

`http://msdn.microsoft.com/library/techart/intromdx.htm`

For an example of a query using an Essbase Report Specification, see Essbase Report Specifications in "Retrieving Data" of the *Developer's Guide*. For specifics, see the online documentation in the Essbase installation directory:

`\docs\techref\RPTIND.HTM`

See Also

"selectableSlicerDimensions" on page 202, DataBlox `generateQuery()` method.

retainSlicerMemberSet

Specifies whether to retain the member selections made in the grid.

Data Sources

Multidimensional

Syntax

`retainSlicerMemberSet="`*`persistMemberSelection`*`"`

where:

Argument	Default	Description
`persistMemberSelection`	`true`	Specify `true` to persist member selections made in the grid.

Usage

When `true` (default) the member selections made in the grid are retained and the page filters will show the children of the selected member. When `false`, the member selections made in the grid are not retained when users move a dimension back and forth between the Page dimension and the Row or Column dimension.

rowSort

Specifies how to sort data values for members on the row axis.

Data Sources

All

Syntax

`rowSort="`*`sortString`*`"`

where:

Argument	Default	Description
`tuple`	none	The tuple on the row axis that specifies the row to be sorted.
`dimension`	none	The dimension on the column axis for which grouping is preserved. Specify `null` to indicate no grouping is to be preserved on the column axis.
`ascending`	none	Specify `true` to sort ascending; `false` to sort descending.
`preserveHierarchy`	`false`	Specify `true` to preserve the hierarchy in the column axis, keeping members with their parents after the sort operation; `false` to not preserve hierarchy. This argument is only valid in the Java method.

Argument	Default	Description
sortString	none	A comma-delimited string in one of the following formats: • *tupleIndex*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction* • *tupleIndex*, *groupingNestLevel*, *direction*, *preserveHierarchy* *tupleIndex* — string representation of an integer representing the zero-based tuple index member (row) to sort, where 0 indicates the topmost row. *groupingNestLevel* — string representations of an integer representing the dimension on the column axis for which grouping is preserved. For example, if Time and Measures are on the column axis, a value of 1 sorts into sequence within the Measures dimension. Specify -1 to sort without regard to column groupings. The default is -1. *direction* — a case-insensitive string of either "Ascending", "Asc", "Descending" or "Desc". *preserveHierarchy* — string representation of a boolean. See the preserveHierarchy argument. Defaults to false. For example: `setRowSort("1,0,asc");` `setRowSort("1,0,asc,true");` `setRowSort("0,descending");`

Usage
The getRowSort method returns a string of four comma-separated items: *tupleIndex*, *groupingNestLevel*, *direction*, and *preserveHierarchy.*

Examples
The following example demonstrates the use of the rowSort tag attribute:

```
rowSort="1, 0, asc"
```

See Also
"columnSort" on page 179, DataBlox removeRowSort() method in the Javadoc documentation.

schema
Specifies the name of the schema to access.

Data Sources
All

Syntax
schema="*schema*"

where:

Argument	Default	Description
schema	empty string	Name of the schema

Usage
The value for schema is one of the values provided when defining a data source to DB2 Alphablox. If you do not specify the schema property for a DataBlox, the value is taken from the data source definition.

A schema is a "database" in IBM DB2 OLAP Server or Hyperion Essbase terminology.

See Also
"catalog" on page 178

selectableSlicerDimensions
Specifies the dimensions that appear on the page (slicer) axis. The slicer dimensions act as filters on the data.

Data Sources
Multidimensional

Syntax
```
selectableSlicerDimensions="dimensionString"
```

where:

Argument	Default	Description
dimensionString	empty string	A comma-separated string of unique dimension names.

Usage
The selectableSlicerDimensions property has no effect on dimensions that already reside on the row or column axis. It can only operate on dimensions that are currently on the "Other" (unused) axis. This method is relevant for multidimensional data sources only.

showSuppressDataDialog

When the useOlapDrillOptimization property is set to true, if either suppressMissingColumns or suppressMissingRows is also set to true, this property specifies whether a warning dialog should pop up. This dialog alerts users of the possibility of incomplete data when they drill down and then perform further data analysis operations.

Data Sources
Microsoft Analysis Services, SAP BW

Syntax
```
showSuppressDataDialog="showDialog"
```

where:

Argument	Default	Description
showDialog	true	Specify true to pop up an alert dialog; false to suppress the pop-up dialog.

Usage
When the property (set either by assemblers through Blox tags, a Java method or by users through the Blox user interface) and the useOlapDrillOptimization property are both set to true, users may only see partial data. This happens when they drill down and then perform other actions such as changing page filters, drilling up, or using Member Filter. When this sequence of user actions occur, a dialog will pop up that alerts the users of this possibility and recommends the users to turn off Suppress Missing. When this showSuppressDataDialog property is set to false, the dialog will not pop up.

See Also
"useOlapDrillOptimization" on page 208, "suppressMissingColumns" on page 203, "suppressMissingRows" on page 204

suppressDuplicates
Specifies whether to remove from the grid those rows or columns containing duplicate header values.

Data Sources
Multidimensional

Syntax
suppressDuplicates="*suppress*"

where:

Argument	Default	Description
suppress	true	Specify true to suppress duplicate header values; false to leave them.

Usage
To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the SUPSHARE command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also
"suppressMissingColumns" on page 203, "suppressMissingRows" on page 204, "suppressNoAccess" on page 205, "suppressZeros" on page 205

suppressMissingColumns
Specifies whether to remove from the grid those columns containing no data at all.

Data Sources
Multidimensional

Syntax

```
suppressMissingColumns="suppress"
```

where:

Argument	Default	Description
suppress	false	Specify true to suppress columns containing no data; false to leave them.

Usage

Use the missingValueStringproperty on GridBlox to specify what to display in cells with no value.

When the data source is Microsoft Analysis Services or SAP BW, this property should be used with care in conjunction with the useOlapDrillOptimization property. When both properties are set to true, users may only see partial data when they drill down and then perform other actions such as changing page filters, drilling up, or using Member Filter. See "useOlapDrillOptimization" on page 208 for more information.

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the SUPSHARE command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

"suppressMissingRows" on page 204, "suppressDuplicates" on page 203, "suppressNoAccess" on page 205, "suppressZeros" on page 205

suppressMissingRows

Specifies whether to remove from the grid those rows containing no data at all.

Data Sources

Multidimensional

Syntax

```
suppressMissingRows="suppress"
```

where:

Argument	Default	Description
suppresss	false	Specify true to suppress rows containing no data; false to leave them.

Usage

Use the missingValueString property on GridBlox to specify what to display in cells with no value.

When the data source is Microsoft Analysis Services or SAP BW, this property should be used with care in conjunction with the useOlapDrillOptimization property. When both properties are set to true, users may only see partial data

when they drill down and then perform other actions such as changing page filters, drilling up, or using Member Filter. See “useOlapDrillOptimization” on page 208 for more information.

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the `SUPSHARE` command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

“suppressMissingColumns” on page 203, “suppressMissingRows” on page 204, “suppressDuplicates” on page 203, “suppressNoAccess” on page 205, “suppressZeros” on page 205

suppressNoAccess

Specifies whether to remove from the grid those rows or columns containing data the users cannot access.

Data Sources

Multidimensional

Syntax

```
suppressNoAccess="suppress"
```

where:

Argument	Default	Description
`suppress`	`false`	Specify `true` to suppress data the users cannot access; `false` to leave it.

Usage

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the `SUPSHARE` command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

“suppressDuplicates” on page 203, “suppressMissingColumns” on page 203, “suppressMissingRows” on page 204, “suppressZeros” on page 205

suppressZeros

Specifies whether to remove from the grid those rows or columns containing all zeros.

Data Sources

Multidimensional

Syntax

```
suppressZeros="suppress"
```

where:

Argument	Default	Description
suppress	false	Specify true to suppress columns or rows containing all zeros; false to leave them.

Usage

To suppress duplicate shared members in IBM DB2 OLAP Server or Hyperion Essbase result sets, use the SUPSHARE command in your report script query. To learn more about this command, refer to your IBM DB2 OLAP Server or Hyperion Essbase documentation.

See Also

"suppressDuplicates" on page 203, "suppressMissingColumns" on page 203, "suppressMissingRows" on page 204, "suppressNoAccess" on page 205

textualQueryEnabled

Specifies whether the data query should be restored using the textual query rather than the serialized query.

Data Sources

All

Syntax

```
textualQueryEnabled="textualQuery"
```

where:

Argument	Default	Description
textualQuery	false	Specify whether the data query should be loaded using the query string stored in the textual query file. By default, a bookmark is loaded using the serialized query.

Usage

When a bookmark is first added, the *delta* between the query set in the DataBlox and the query that generates the current data view is saved in a <bookmark_name>.data text file and a <bookmark_name>.query file that stores the query as serialized object. Later when the bookmark is loaded, the serialized query is used unless this property is set to true. This is useful in cases where there is a change of data outline or member names and you want to modify the textual query accordingly. Manipulation of the textual query is more efficient since DB2 Alphablox does not need to manipulate the result set to match the serialized object.

Note, however, that the textual query is not updated when the bookmark is resaved with a different data view. If you anticipate the need to use textual queries, you may want to ensure the textual query is up-to-date. In this case, you can capture a bookmark save event and get the current textual query using the DataBlox generateQuery() method to update the query in the bookmark. This involves the use of the addEventFilter() common Blox method to add a method that implements the BookmarkSaveFilter interface.

See Also

"Serialized Query and Textual Query" on page 53, "Bookmark Events and Event Filters" on page 52, DataBlox `generateQuery()` method in the Javadoc documentation.

useAASUserAuthorizationEnabled

Specifies whether to use the user name and password entered during DB2 Alphablox login for authentication to an IBM DB2 OLAP Server or Hyperion Essbase data source.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase

Syntax

`useAASUserAuthorizationEnabled="`*`useIt`*`"`

where:

Argument	Default	Description
`useIt`	`false`	Specify `true` to use the DB2 Alphablox login information for IBM DB2 OLAP Server or Hyperion Essbase authentication; `false` to use the normal authentication process.

Usage

This property is valid only when using DB2 Alphablox in a standalone configuration without using external web server security.

When set to `true`, the data source uses the values entered when the user logged in for access to the data source. When set to `false`, the data source uses the normal authentication process.

useAliases

Specifies whether to use aliases or database member values in row and column headings.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase

Syntax

`useAliases="`*`useAliases`*`"`

where:

Argument	Default	Description
`useAliases`	`false`	Specify `true` to use aliases; `false` to use member names

Usage

Database member values are typically codes (such as 001-200); aliases are names (such as Diet Cola).

This property overrides the use of the `{OUTALTNAMES}` command in an IBM DB2 OLAP Server or Hyperion Essbase report script.

useOlapDrillOptimization

Specifies whether drill optimization should be enabled for Microsoft Analysis Services and SAP BW data sources.

Data Sources

Microsoft Analysis Services, SAP BW

Syntax

`useOlapDrillOptimization="optimize"`

where:

Argument	Default	Description
`optimize`	`true`	Specify `true` to use the query optimization

Usage

By default, this property is set to `true` for Microsoft Analysis Services or SAP BW data sources for better query performance. Use this property with care in conjunction with the `suppressMissing` property. When both properties are set to `true`, users may only see partial data when they drill down and then perform other actions such as changing page filters, drilling up, or using Member Filter. When this sequence of user actions occur, a dialog will pop up that alerts the users of this possibility and recommends the users to turn off **Suppress Missing**. This dialog can be turned off with the `showSuppressDataDialog` property.

See Also

"showSuppressDataDialog" on page 202, "suppressMissingColumns" on page 203, "suppressMissingRows" on page 204

userName

Specifies the database user name to use when accessing the data source.

Data Sources

All

Syntax

`userName="userName"`

where:

Argument	Default	Description
`userName`	empty string	Database user name.

Usage

A default user name is one of the values provided when defining a data source to DB2 Alphablox. If no `userName` property is specified for a DataBlox, the value is taken from the data source definition.

If you use the associated `setUserName()` method in conjunction with `setDataSourceName()` method, you should set the user name after calling `setDataSourceName()`. Otherwise, the DataBlox will connect with all the properties in the data source specified and override any properties set earlier. This is because

the `setDataSourceName()` method also reads in the properties of the data source such as username, password, catalog, schema, query, and dimensions on page axis. Therefore, if you want to set any of these properties using the Java methods, set them after calling `setDataSourceName()`.

Tip: The order these data source properties are set is not an issue if you use the Blox tags. The tags are designed to enforce that the data source is set before the call to set the other data source properties, the side effect is taken care for you.

See Also

"dataSourceName" on page 183

Chapter 11. DataLayoutBlox Tag Reference

This chapter contains reference material for DataLayoutBlox. For general reference information about Blox, see Chapter 3, “General Blox Reference Information,” on page 15. For information on how to use this reference, see Chapter 1, “Using This Reference,” on page 1.

DataLayoutBlox Overview

DataLayoutBlox provides a graphical representation of multidimensional database dimensions, organized by the axis on which they appear (row, column, or page). A fourth axis, other, lists dimensions that do not appear on any other axis, but are available in the current result set.

Graphical User Interface

The DataLayoutBlox GUI allows users to perform the following tasks:
- view and move dimensions between axes
- drag-and-drop dimension into or out of GridBlox
- access the Member Filter

DataLayoutBlox has two interface types: tree (default) and drop list. The tree interface contains expandable and collapsible tree menus and supports drag-and-drop operations. The drop list interface uses drop lists to support moving dimensions among axes.

For instructions on using the DataLayoutBlox user interface, see DataLayoutBlox user help. You can access the user help by clicking the help button on the toolbar in the Blox user interface.

DataLayoutBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a DataLayoutBlox. For a copy and paste version of the tag with all the attributes, see “DataLayoutBlox JSP Custom Tag” on page 462.

```
<blox:dataLayout
       [attribute="value"] >
</blox:dataLayout>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
applyPropertiesAfterBookmark

Attribute
bloxEnabled
bloxName
bookmarkFilter
height
helpTargetFrame
hiddenDimensionsOnOtherAxis
interfaceType
maximumUndoSteps
noDataMessage
render
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:dataLayout>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
width="650" />
```

Examples

```
<blox:dataLayout
     id="namedDataLayoutBlox"
     width="100"
     height="400" />
```

DataLayoutBlox Tag Attributes

This section lists the tag attributes available through DataLayoutBlox in alphabetical order. For complete descriptions of common Blox properties, see “Tag Attributes Common to Multiple Blox” on page 30. For reference information on DataLayoutBlox methods see the DataLayoutBlox class in the com.alphablox.blox package in the Javadoc documentation.

id

This is a common Blox property. For a complete description, see “id” on page 36.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see “applyPropertiesAfterBookmark” on page 30.

bloxEnabled

This is a common Blox property. For a complete description, see “bloxEnabled” on page 32.

bloxName

This is a common Blox property. For a complete description, see “bloxName” on page 32.

bookmarkFilter

This is a common Blox property. For a complete description, see “bookmarkFilter” on page 31.

height

This is a common Blox property. For a complete description, see “height” on page 35.

helpTargetFrame

This is a common Blox property. For a complete description, see “helpTargetFrame” on page 35.

hiddenDimensionsOnOtherAxis

Specifies the dimensions to be hidden on the Other axis in the Data Layout panel.

Data Sources

Multidimensional

Syntax

```
hiddenDimensionsOnOtherAxis="dimensionsToHide"
```

where:

Argument	Default	Description
`dimensionsToHide`	null	A comma-delimited String representing the dimensions to hide on the Other axis.

Usage

By default, any dimensions in the cube not specified in the data query are placed in the Other axis in the Data Layout panel. By hiding the dimensions in the Other axis, you can prevent users from pivoting these dimensions or changing the currently selected member for these dimensions. For example, you may want to hide IBM DB2 OLAP Server or Hyperion Essbase Attribute dimensions or prevent your user from changing the Measures filter. Or in your Microsoft Analysis Services data source, you may have multiple hierarchies [Time].[Fiscal] and [Time].[Calendar]. Your query already specifies to have [Time].[Fiscal] on the Column axis and need to hide [Time].[Calendar] in the Other axis to avoid confusion.

The dimensions are only hidden on the UI; they still exist in the data. If a dimension is specified in your query to appear, for example, on the Column axis and you hide it again on the Other axis, the dimension will still show on the Column axis. However, once the users pivot the dimension into the Other axis, it will become hidden from then on. To unhide a dimension, the `hiddenDimensionsOnOtherAxis` property needs to be reset.

Examples

The following example hides the [Time].[Calendar], Measures, and [Attribute Calcs] dimensions from on the Other axis.

```
hiddenDimensionsOnOtherAxis="[Time].[Calendar], Measures,
[Attribute Calcs]"
```

interfaceType

Sets the interface type for the Data Layout panel.

Data Sources

Multidimensional

Syntax

```
interfaceType="type"
```

where:

Argument	Default	Description
type	tree	Valid values are tree and dropList. When set to tree, the Data Layout panel has a tree navigation interface with tree menus that expand and collapse. See "DataLayoutBlox Overview" on page 211.

maximumUndoSteps

This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 36.

noDataMessage

This is a common Blox property. For a complete description, see "noDataMessage" on page 37.

render

This is a common Blox property. For a complete description, see "render" on page 38.

visible

This is a common Blox property. For a complete description, see "visible" on page 40.

width

This is a common Blox property. For a complete description, see "width" on page 40.

Chapter 12. GridBlox Tag Reference

This chapter contains reference material for GridBlox tags and tag attributes. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

GridBlox Overview

The user interface for GridBlox consists of a tabular data display area consisting of rows and columns of data cells and optional grid controls. Users can view multidimensional data on row, column, and page axes. Users can manipulate the data presentation by drilling up and down through data hierarchies, move data dimensions to different axes, include or omit data dimensions, and so forth.

GridBlox displays data from both relational and multidimensional data sources. It displays relational data in a two-dimensional row-and-column format. It displays multidimensional data in an interactive, multidimensional grid format, enabling users to perform multidimensional analysis. DB2 Alphablox includes a cube server that transforms relational data into multidimensional cubes, enabling GridBlox to display the data in multidimensional format.

The DHTML client displays a GridBlox as an HTML element with the `id` specified in the `<blox:grid>` tag. Each grid in the HTML element is a `DIV` element, and has all the attributes, methods and events typically associated with a `DIV`. In addition, a `Selection` object is available that represents the cells that are currently selected. For details on the DHTML Client API, see the *Developer's Guide*.

GridBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a GridBlox. For a copy and paste version of the tag with all the attributes, see "GridBlox JSP Custom Tag" on page 462.

Syntax

```
<blox:grid
   [attribute="value"] >
      [<blox:cellAlert
           [attribute="value"] />]
      [<blox:cellEditor
           [attribute="value"] />]
      [<blox:cellFormat
           [attribute="value"] />]
      [<blox:cellLink
           [attribute="value"] />]
      [<blox:drillThroughWindow
           [attribute="value"] />]
      [<blox:editableCellStyle
```

```
          [attribute="value"] />]
     [<blox:editedCellStyle
          [attribute="value"] />]
     [<blox:formatMask
          [attribute="value"] />]
     [<blox:formatName
          [attribute="value"] />]
</blox:grid>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

and where the attributes are one of the following:

<blox:grid> tag
Attribute
id
autosizeEnabled
applyPropertiesAfterBookmark
bandingEnabled
bloxEnabled
bloxName
bookmarkFilter
columnHeadersWrapped
columnWidths
commentsEnabled
defaultCellFormat
drillThroughEnabled
drillThroughWindow
editableCellStyle
editedCellStyle
enablePoppedOut
expandCollapseMode
gridLinesVisible
headingIconsVisible
headingsEnabled
height
helpTargetFrame
informationWindowName
maximumUndoSteps
menubarVisible
missingValueString
noAccessValueString
noDataMessage
poppedOut

<blox:grid> tag
Attribute
poppedOutHeight
poppedOutTitle
poppedOutWidth
relationalRowNumbersOn
removeAction
render
rightClickMenuEnabled
rowHeadersWrapped
rowHeadingsVisible
rowHeadingWidths
rowHeight
rowIndentation
showColumnDataGeneration
showColumnHeaderGeneration
showRowDataGeneration
showRowHeaderGeneration
toolbarVisible
visible
width
writebackEnabled

<blox:cellAlert> nested tag **See "cellAlert" on page 223.**
Attribute
index
apply
background
condition
description
enabled
font
foreground
format
group
link
image_align
image
scope
value

<blox:cellAlert> nested tag **See "cellAlert" on page 223.**
Attribute
value2

<blox:cellEditor> nested tag **See "cellEditor" on page 229.**
Attribute
index
scope

<blox:cellFormat> nested tag **See "cellFormat" on page 231.**
Attribute
index
background
font
foreground
format
group
scope

<blox:cellLink> nested tag **See "cellLink" on page 235.**
Attribute
index
description
link
scope
image_align
image

<blox:drillThroughWindow> nested tag **See "drillThroughWindow" on page 242.**
Attribute
height
locationbarVisible
menubarVisible
name
resizable

<blox:drillThroughWindow> nested tag **See "drillThroughWindow" on page 242.**
Attribute
scrollbarsVisible
statusbarVisible
toolbarVisible
url
width

<blox:editableCellStyle> nested tag **See "editableCellStyle" on page 244.**
Attribute
background
font
foreground

<blox:editedCellStyle> nested tag **See "editedCellStyle" on page 245.**
Attribute
background
font
foreground

<blox:formatMask> nested tag **See "formatMask" on page 246.**
Attribute
index
mask

<blox:formatName> nested tag **See "formatName" on page 248.**
Attribute
index
name

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

When there are no nested tags (such as the <blox:cellAlert> or <blox:cellStyle> tag), you can substitute the closing </blox:grid> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
width="650" />
```

When there are nested tags, the shorthand notation is not valid and a closing tag is required.

Examples

```
<blox:grid id="myGrid"
     height="400"
     width="500"
     bandingEnabled="true" />
<blox:grid id="anotherGrid"
     height="300"
     width="500"
     bandingEnabled="true">
     <blox:cellAlert index="1"
          condition="any"
          background="cyan" />
</blox:grid>
```

GridBlox Tag Attributes By Category

This section lists the tag attributes unique to GridBlox. For tag attributes common to several Blox, see the "Common Blox Tag Attributes by Category" on page 29. The tag attributes supported by GridBlox are organized in the cross reference as follows:

For GridBlox methods, see the GridBlox class in the com.alphablox.blox package in the Javadoc. For its client-side API for the DHTML client, see *Developer's Guide*.

Grid Appearance

The following tag attributes affect how the grid appears on a page.

- bandingEnabled
- cellLink
- columnHeadersWrapped
- columnWidths
- expandCollapseMode
- gridLinesVisible
- headingIconsVisible
- informationWindowName
- menubarVisible
- missingValueString
- noAccessValueString
- relationalRowNumbersOn

- removeAction
- rightClickMenuEnabled
- rowHeadersWrapped
- rowHeadingsVisible
- rowHeadingWidths
- rowHeight
- rowIndentation
- showColumnDataGeneration
- showColumnHeaderGeneration
- showRowDataGeneration
- showRowHeaderGeneration

Numeric Formatting

These tag attributes define how numeric values appear in the grid data area. For details on available formats, see:

http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html

- cellFormat
- defaultCellFormat
- formatMask
- formatName

Cell Alerts

The following table shows tag attributes associated with cell alerts:

- cellAlert

Drill to Relational Detail

Drillthrough operations are supported for IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services data sources which have drillthrough reports set up through Essbase Integration Services (EIS). This feature is also supported for Microsoft Analysis Services.

- drillThroughEnabled
- drillThroughWindow

Printing

Printing tag attributes apply to the grid when it is rendered for delivery in print format.

- defaultCellFormat
- headingsEnabled

Grid UI for Writeback and Comments

The Writeback UI tag attributes set conditions on the GridBlox data area that permit users to change data cell values. The Comments UI tag attribute (commentsEnabled) specifies whether 1) the menu items for adding and displaying comments are displayed in the Grid cell right-click menu and 2) whether to display the comment indicator on the upper right corner when cell comments are available. These properties are used with "GridBlox Tag Attributes" on page 222.

- cellEditor
- commentsEnabled

- editableCellStyle
- editedCellStyle
- writebackEnabled

Popped Out Properties

The following table lists the tag attributes regarding displaying GridBlox in a separate, popped out browser window.

- enablePoppedOut
- poppedOut
- poppedOutHeight
- poppedOutTitle
- poppedOutWidth

GridBlox Tag Attributes

This section describes the tag attributes supported by GridBlox in alphabetical order. For reference information on GridBlox methods, see the GridBlox class in the com.alphablox.blox package in the Javadoc. Common Blox properties available from GridBlox are listed but not described. For complete descriptions of common Blox tag attributes, see "Tag Attributes Common to Multiple Blox" on page 30.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 36.

applyPropertiesAfterBookmark

This is a common Blox property. For a detailed description, see "applyPropertiesAfterBookmark" on page 30.

autosizeEnabled

Specifies whether columns should automatically resize to accommodate the largest data value.

Data Sources

All

Syntax

```
autosizeEnabled="autosize"
```

where:

Argument	Default	Description
autosize	true	Specify false to render columns and rows based on columnWidths, rowHeadingWidths and rowHeight settings.

Examples

```
autosizeEnabled = "true"
```

bandingEnabled

Specifies whether to enable alternate background colors for grid rows.

Data Sources
All

Syntax
```
bandingEnabled="enable"
```

where:

Argument	Default	Description
enabled	true	Specify true to enable cell banding; false to disable it.

Usage
The default is true when the application's default render mode is set to DHTML.

Examples
```
bandingEnabled = "true"
```

bloxEnabled
This is a common Blox property. For a complete description, see "bloxEnabled" on page 32.

bloxName
This is a common Blox property. For a complete description, see "bloxName" on page 32.

bookmarkFilter
This is a common Blox property. For a complete description, see "bookmarkFilter" on page 31.

cellAlert
Specifies a rule for highlighting values in numeric data cells.

Data Sources
All

Syntax
```
<blox:cellAlert
      index="cellAlertNumber"
      apply="row|column|cell"
      background="background"
      condition="condition"
      description="description"
      enabled="enabled"
      font="font"
      foreground="foreground"
      format="formatmask"
      group="groupName"
      image="image"
      image_align="left|right|center"
      link="link"
      scope="scope"
      value="value1"
      value2="value2"
/>
```

where:

Attribute	Required?	Description
apply	No	The area of the grid subject to highlighting. Possible values: • ROW: Highlight the entire row if any cell within the row meets the condition. The first row-applied alert takes precedence over other row-applied alerts. • COLUMN: Highlight the entire column if any cell within the column meets the condition. The first column-applied alert takes precedence over other column-applied alerts. • CELL: Highlight only the specific cells that meet the condition (the default). A cell-applied alert will override a row- or column-applied alert.
background	No	The cell's background color. Use a color name or hexadecimal value.
condition	Yes	A condition to apply to the cell value. Cells that meet the condition activate the alert. Possible values: • ANY: Accept cells of any value. • MISSING: Accept only cells with missing values. • NA: Accept only cells where the value is not available. The following possible values work together with the value and value2 attributes: • LT or < • GT or > • EQ or = • LTEQ or <= • GTEQ or >= • BETWEEN (*value*, *value2*): The value is greater than or equal to *value* and less than or equal to *value2*.
description	No	A description of the cell alert. This description appears as a tool tip when the user hovers the mouse pointer over the cell.
enabled	No	Specifies whether the cell alert is active. Set to false to disable a cell alert. The default is true.
font	No	The *font name:style:point* use for the cell's text. • *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace. • *style*: Valid font styles are: plain, italic, bold, and bolditalic • *point*: An integer for point size (usually 8-36). If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. For examples: font="Arial:bolditalic:12" font=":Bold:12"
foreground	No	The cell's text color. Use a color name or hexadecimal value.
format	No	Format mask to apply to cell data. For more information, see "defaultCellFormat" on page 240.
group	No	The name of a group of cell alerts to be treated as a traffic lighting group. Cell alerts with the same group name will show up as one set of traffic lights in the traffic lights creation and management user interface.

Attribute	Required?	Description
image	No	A custom image that points to the defined link. If you define a link but do not specify a custom image, a default image information image is displayed. The URL to the image can be either absolute or relative: • For absolute URLs, the string should begin with "http://". • For relative URLs: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. – Starting the string without a slash indicates that the URL is relative to the current document.
image_align	No	Sets the position of the custom image specified by the image attribute. Valid values are: • LEFT: Place image before the cell contents (default) • RIGHT: Place image after the cell contents
link	No	A hyperlink for HTML rendering or a call to a JavaScript method. When the link is to a URL, it always opens in a new window. The following entities in the link value are replaced by the indicated values when the link is generated. Entities must start with an ampersand (&) and end with a semicolon (;). • &Description; — the cell alert description • &Value; — the cell value • &ColHeader; — ampersand-separated dimension/member value pairs • &RowHeader; — ampersand-separated dimension/member value pairs • &ColIndex; — the display index (0-based) of the column in the grid • &RowIndex; — the display index (0-based) of the row in the grid In the following example, the link for the cell alert is set to: `link="decoderequest.jsp?row=&RowHeader;&column=` `&ColHeader;&value=&Value;&rowIndex=&RowIndex;&col` `Index=&ColIndex;"` If the cell with the value of 1234.55 on the third row of the Time dimension and the fourth column of the Product dimension is clicked on, the URL passed through will be: `decoderequest.jsp?row=Time=Q3&column=Product=Chocolate%20` `Nuts&value=1234.55&rowIndex=2&colIndex=3`

Attribute	Required?	Description
scope	No	The cells to which the alert should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). For MSAS and Alphablox Cube Server data sources, use unique names starting with the cube name to ensure the correct member at the correct level is found. For SAP BW, always use unique names starting with the query name. Using display names will result in data exception or random data. The scope applies to all axes of the result set, not just the row and column axes. Specify the scope as follows: `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for Essbase: `scope={Product:Coke} {Scenario: Actual, Budget}` For MSAS, Alphablox Cube Server, and SAP BW data sources: `scope="{[My Cube].[Product]:[Code]} {[My Cube].[Scenario]: [Actual], [Budget]}"` For relational data sources, use "{columns: *columnName* [,*columnName2*,...]}". The following member search functions are available for specifying the level of members the cell alert should be applied to: • Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:leaf(East)}" (Essbase) • Child(): the children of the specified member. Only one member can be specified in the function. Example: scope="{Market:child(East)}" (Essbase) • Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: scope="{[My Cube].[Market]:descendants([East])}" (MSAS, Alphablox Cube, SAP BW) • Gen(): all members of the specified generation. Example: scope="{Market:gen(2)}" (Essbase) • Not(): members to which the cell alert should not be applied. You can specify multiple members, separated by a comma. Example: scope="{[My Cube].[Market]:not([East], [West])}" (MSAS, Alphablox Cube, SAP BW) The function names are case-insensitive. You can combine the functions in the scope statement.
value	No	The value to compare when condition is LT, GT, EQ, LTEQ, GTEQ, or the lesser value when condition is BETWEEN.
value2	No	The greater value when condition is BETWEEN.
index	No	The number of the cellAlert to define. If you do not specify this attribute, the next available cellAlert number is used. For instance, if cellAlerts 1-4 are already defined, cellAlert 5 is used.

Usage

Cell alert formatting applies only to numerical content. The number of the cell alert dictates the order in which it is evaluated. Each data cell value is evaluated against all defined alerts, starting with the first cellAlert (index="1"), to the highest defined alert number. The first cell alert that matches the data cell's value and scope is the only alert applied to that cell. Be sure to define cell alerts in the appropriate order if there are overlaps.

- If you do not specify a given attribute, its default value applies to the cell. For example, if the default cell background is white and you do not specify a background attribute, the cell background remains white.
- For editable cells, cell alert formatting takes precedence over cell editor formatting when a grid first appears. Once a cell is edited, cell editor color settings take precedence over those specified by cell alerts.
- Use unique names (base name in IBM DB2 OLAP Server or Hyperion Essbase) or display names for the dimension and member name string specified in the scope. Using unique names allows you to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, you can specify a member, regardless of the alias table in use, by using the base name.
- For MSAS and Alphablox Cube Server data sources, when specifying the scope:
  - Enclose dimension and member names in brackets, starting with the cube name. For example, ″[My cube].[Market].[West].[California]″.
  - If a member name is not enclosed in brackets, it will be treated as a member whose name is all uppercase.

Examples

- Divide a value by 1000 before displaying it:

```
<blox:cellAlert index="1"
                condition="any"
                format="#,###/1000;[red](#,###/1000)"
                background="#3333FF"
                scope="{Scenario: Budget}" />
```

  For MSAS, Alphablox Cube, and SAP BW, the scope should be `scope="{[My Cube].[Scenario]: [Budget]}`.

- Use a symbol (in this case, the percent sign) as a literal character:

```
<blox:cellAlert index="3"
                condition="any"
                format="#'%';[red](#'%')"
                background="#9999FF"
                scope="{Scenario: Variance %}" />
```

  For MSAS, Alphablox Cube, and SAP BW, the scope should be `scope="{[My Cube].[Scenario]: [Variance %]}`.

- Multiply a value by 100 before it is displayed and use Times New Roman, boldface, size 14 for font:

```
<blox:cellAlert index="4"
                condition="any"
                format="#.##*100%;[red](#.##*100%)"
                background="#CCCCFF"
                scope="{Scenario: Variance %}"
                font="Times New Roman:bold:14" />
```

- The following example sets cell alert 2 on a grid displayed in a PresentBlox. The alert tests the Central member of the Market dimension for values greater than 1000. When a data value meets this criterion:
  - The foreground (font) color becomes green.
  - The background color becomes white.
  - Values are aligned on the right.

```
PresentBlox.getGridBlox().setCellAlert(2, "condition=GT, value=1000,
scope={Market:Central}, foreground=green, background=white, align=right");
```

  For MSAS and SAP BW, the scope should be `scope="{[My Cube].[Market]: [Central]}`.

- The following example highlights all numerical content with a cyan background:

```
<blox:cellAlert index="1"
                condition="any"
                background="cyan" />
```

  This will apply a cyan background to all cells that contain numerical data. If a cell contains no data, this background color will not be applied.

- A series of related cell alerts paints increasing cell amounts in different colors:

```
<blox:cellAlert index="1"
                condition="Between" value="1000" value2="3000"
                format="00.00"
                foreground="Orange"
                scope="{[My Cube].[Market]:[East],[West],[South],[Central]}" />

<blox:cellAlert index="2"
                condition="Between" value="3001" value2="5000"
                scope="{[My Cube].[Year]:[Qtr1],[Qtr2]}"
                format="00.00"
                foreground="Blue" />

<blox:cellAlert index="3"
                condition="GT"
                value="5000"
                format="00.00"
                foreground="Magenta" />

<blox:cellAlert index="4"
                condition="LT"
                value="1000"
                format="(00.00)"
                foreground="Red" />
```

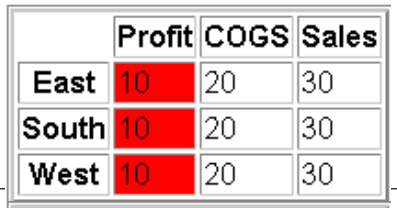
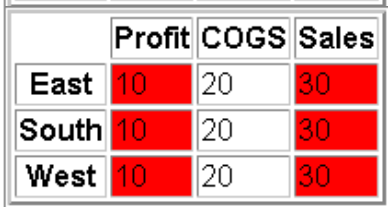
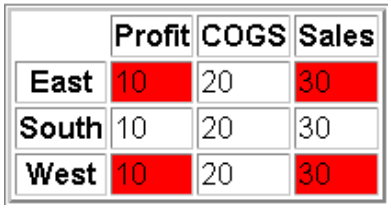
and results in the following grid:

Year	Product	East	West	South	Central	Market
Qtr1	Colas	2747.00	1042.00	1051.00	2208.00	7048.00
	Root Beer	(562.00)	2325.00	1465.00	2369.00	6721.00
	Cream Soda	(591.00)	2363.00	(561.00)	2414.00	5929.00
	Fruit Soda	1480.00	1407.00		2118.00	5005.00
	Diet Drinks	(555.00)	2025.00	1146.00	3291.00	7017.00
	Product	5380.00	7137.00	3077.00	9109.00	24703.00
Qtr2	Colas	3352.00	(849.00)	1198.00	2473.00	7872.00
	Root Beer	(610.00)	2423.00	1540.00	2457.00	7030.00
	Cream Soda	(922.00)	2739.00	(529.00)	2579.00	6769.00
	Fruit Soda	1615.00	1504.00		2317.00	5436.00
	Diet Drinks	(652.00)	1975.00	1289.00	3420.00	7336.00
	Product	6499.00	7515.00	3267.00	9826.00	27107.00

Scope Examples

The examples in the following table assume `background=red` and `condition=any`, and show the result using different scopes.

Note: The `scope` attribute applies to all axes of the result set, not just the row and column axes. For example, if the grid is filtered on the profit member of the accounts dimension (that is, the accounts dimension is placed on the page filter with the profit member selected), the grid appears as shown in the first example below.

scope Example	Result
The following scope applies only to the profit member of the accounts dimension: `scope="{Accounts: Profit}"`	
The following scope applies to the profit and sales members of the accounts dimension: `scope="{Accounts: Profit, Sales}"` Note that this means profit OR sales. You can also specify the scope as follows: `scope="{Accounts: not(COGS)}"`	
The following scope applies where the profit and sales members of the accounts dimension intersect with the east and west members of the market dimension. This is an AND operation. `scope="{Accounts: Profit, Sales}, {Market: East, West}"` or `scope="{Accounts: not(COGS)}, {Market: not(South)}"`	

	Profit	COGS	Sales
East	10	20	30
South	10	20	30
West	10	20	30

	Profit	COGS	Sales
East	10	20	30
South	10	20	30
West	10	20	30

	Profit	COGS	Sales
East	10	20	30
South	10	20	30
West	10	20	30

See Also

"cellEditor" on page 229, "cellFormat" on page 231, "cellLink" on page 235, the `listCellAlertIds()` method in the Blox API Javadoc, "Cell Alerts" on page 221

cellEditor

Specifies a rule for defining and highlighting an editable area of data cells.

Data Sources

All, except SAP BW

Syntax

```
<blox:cellEditor
     index="cellEditorNumber"
     scope="scope" >
</blox:cellEditor>
```

where:

Attribute	Required?	Description
scope	Yes	The cells to which the editor should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). For MSAS and Alphablox Cube Server data sources, use unique names starting with the cube name to ensure the correct member at the correct level is found. The scope applies to all axes of the result set, not just the row and column axes. Specify the scope as follows: `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for Essbase: `scope="{Product:Coke} {Scenario: Actual, Budget}"` For MSAS and Alphablox Cube Server: `scope="{[My Cube].[Product]:[Code]}` `{[My Cube].[Scenario]:[Actual],[Budget]}"` • Enclose dimension names and member names in brackets, starting from the cube name. For example, "[My Cube].[Market].[West].[California]". • If the member name is not enclosed in brackets, it will be treated as a member whose name is all uppercase. Member search functions are available for specifying the level of members the editor should be applied to: • Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: `scope="{Market:leaf(East)}"` (Essbase) • Child(): the children of the specified member. Only one member can be specified in the function. Example: `scope="{Market:child(East)}"` (Essbase) • Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: `scope="{[My Cube].[Market]:descendants([East])}"` (MSAS) • Gen(): all members of the specified generation. Example: `scope="{Market:gen(2)}"` (Essbase) • Not(): members to which the cell editor should not be applied. You can specify multiple members, separated by a comma. Example: `scope="{[My Cube].[Market]:not([East], [West])}"` (MSAS) The function names are case-insensitive. You can combine the functions in the scope statement. See "Scope Examples" on page 228 for more examples.
index	No	Note: This attribute is only valid as a cellEditor tag attribute. For the setCellEditor Java method, use the id argument instead. The number of the cell editor to define. If you do not specify this attribute, the next available cell editor number is used. For example, if cell editors 1-4 are already defined, cell editor 5 is used.

Usage

Use the `scope` attribute to define areas of editable cells in the grid.

To activate cell editors you must set writebackEnabled to `true`.

You can make non-numeric cells to be editable on the grid user interface. However, non-numeric values are written back as missing values. Unlike IBM DB2 OLAP Server or Hyperion Essbase, which only allows writeback of numeric values, Microsoft Analysis Services allows writeback of any data type. Care should be taken when specifying the scope so non-numeric cells are not overwritten with the `"#MISSING"` string. With relational data sources, DB2 Alphablox does not perform data writeback. You need to programmatically get the list of changed cells and their new values and write back the values using JDBC. The benefit of this approach is that you can write back non-numeric data.

Cell alert formatting takes precedence over cell editor formatting when a grid first appears. Once a cell is edited, cell editor color settings take precedence over those specified by cell alerts.

Examples

```
<blox:cellEditor
     index="3"
     scope="scope={[My Cube].[Market]: [East]}" />
```

The following example sets cell editor 2 on a grid displayed in a PresentBlox. Any values within the Market dimension except those for the Central member are editable.

```
<blox:present id="myPresent" >
   <blox:data bloxRef="aPreDefinedDataBlox" />
   <blox:grid>
      <blox:cellEditor
         index="2"
         scope="scope={[My Cube].[Market]:not([Central])" />
   </blox:grid>
</blox:present>
```

See Also

"cellAlert" on page 223, "cellFormat" on page 231, "cellLink" on page 235,
GridBlox `clearCellEditors()` and `listCellEditorIds()` methods in the Javadoc

cellFormat

Specifies the format for data values in the grid's cells.

Data Sources

All

Syntax

```
<blox:cellFormat
      index="cellFormatNumber"
      background="background"
      font="font"
      foreground="foreground"
      format="formatmask"
      group="group"
      scope="scope" >
</blox:cellFormat>
```

where:

Attribute	Required?	Description
background	No	The cell's background color. Use a color name or hexadecimal value.
font	No	The font name:style:point to use for the cell's text. • *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace. • *style*: Valid font styles are: plain, italic, bold, and bolditalic • *point*: An integer for point size (usually 8-36). If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. For examples: font="Arial:bolditalic:12" font=":Bold:12"
foreground	No	The cell's text color. Use a color name or hexadecimal value.
format	Yes	Format mask to apply to cell data. For more information, see "defaultCellFormat" on page 240.
group	No	A name space to group cell formats of the same name as a set.

Attribute	Required?	Description
scope	Yes	The cells to which the format should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). For MSAS and Alphablox Cube Server data sources, use unique names starting with the cube name to ensure the correct member at the correct level is found. For SAP BW, always use unique names starting with the query name. Using display names will result in data exception or random data. The scope applies to all axes of the result set, not just the row and column axes. Specify the scope as follows: `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for Essbase: `scope="{Product:Coke} {Scenario: Actual, Budget}"` For MSAS, Alphablox Cube Server, and SAP BW: `scope="{[My Cube].[Product]:[Code]}` `{[My Cube].[Scenario]: [Actual], [Budget]}"` For relational data sources, use "{columns: *columnName* [,*columnName2*,...]}". The dimension name should be enclosed in brackets and the cube name (MSAS, Alphablox Cube Server) or query name (SAP BW) needs to be included. See the notes in the Usage section below for details on how to specify the dimension and member names. Member search functions are available for specifying the level of members the format should be applied to: • Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:leaf(East)}" (Essbase) • Child(): the children of the specified member. Only one member can be specified in the function. Example: scope="{Market:child(East)}" (Essbase) • Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: scope="{[My Cube].[Market]:descendants([East])}" (MSAS) • Gen(): all members of the specified generation. Example: scope="{Market:gen(2)}" (Essbase) • Not(): members to which the cell format should not be applied. You can specify multiple members, separated by a comma. Example: scope="{[My Cube].[Market]:not([East], [West])}" (MSAS) The function names are case-insensitive. You can combine the functions in the scope statement. See "Scope Examples" on page 228 for the more examples.

Attribute	Required?	Description
index	No	Note: This attribute is only valid as a cellFormat tag attribute. For the setCellFormat Java method, use the id argument instead. The number of the cell format to define. If you do not specify this attribute, the next available cell format number is used. For example, if cell formats 1-4 are already defined, cell format 5 is used.

Usage

The cellFormat tag attribute can be used in conjunction with cell alerts.

Note the following about cell formatting:

- The cell format displayed in the Blox user interface is dependent on the client's locale. It uses the first language in the browser's language list.
- If you do not specify a given property, its default value applies to the cell.
- Do not use the backslash escape character (\) in the format string.
- To display a symbol such as the percent sign (%), use single quotes. If the symbol to display is a double quote, then precede it with a backslash escape character (\).
- The cell format number (*N*) dictates the order in which the format mask is evaluated. Each data cell value is evaluated against all defined masks in order starting from cellFormat1. If a later cellFormat overlaps with an earlier one, the last one is applied. This means the cell format with largest id (Java method) or index (JSP tag) wins. Be sure to define cell format masks in the correct order if there are overlaps.
- Unique names (base names in IBM DB2 OLAP Server or Hyperion Essbase) or display names can be used for the dimension and member name string specified in the scope. Using unique names allows you to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, you can specify a member, regardless of the alias table in use, by using the base name.
- For MSAS and Alphablox Cube Server data sources, when specifying the scope:
  - Enclose dimension and member names in brackets, starting with the cube name. For example, "[My cube].[Market]".
  - If a member name is not enclosed in brackets, it will be treated as a member whose name is all uppercase.
- For IBM DB2 OLAP Server or Hyperion Essbase data sources, use defaultCellFormat or cellFormat to control the formatting of data values instead of the {DECIMAL} report script command.

Examples

- For all cell values in the Profit member of the Accounts dimension and the TV and Video members of the Product dimension, apply a format with two decimal places and a comma separating the hundreds and thousands positions. Show the cell value in red if it is 999.99 or less:

```
<blox:cellFormat
                  index="1"
                  format="#,###.##; [red]###.##"
                  scope="{Accounts:Profit}{Product:TV, Video}" />
```

- For all cell values in the Accounts dimension except the COGS member, apply the thousands format:

```
<blox:cellFormat
                  index="2"
                  format="#,###K"
                  scope="{Accounts:not(COGS)}" />
```

- For all cell values in the Total member of the Accounts dimension, apply a format that includes a dollar sign and rounds to the whole dollar with a specific background color and font style for the cells:

```
<blox:cellFormat
                index="4"
                format="$#,##0"
                scope="{Accounts:Total}"
                font="Arial:Bold:20"
                background="#CCCCFF" />
```

- Format all cell values in the COGS and Total members of the Accounts dimension as two-decimal percentages, and if the percentage is less then 1%, show the number as, for example, 0.55 % rather than .55%:

```
<blox:cellFormat
                index="5"
                format="0.##'%'"
                scope="{Accounts:COGS, Total}" />
```

See Also

"defaultCellFormat" on page 240, "cellAlert" on page 223, "cellEditor" on page 229, "cellLink" on page 235, GridBlox `listCellFormatIds()` method

cellLink

Specifies a rule for defining cells that contain a link.

Data Sources

All

Syntax

```
<blox:cellLink
      index="cellLinkNumber"
      description="description"
      image="image"
      image_align="left|right|center"
      link="link"
      scope="scope" >
</blox:cellLink>
```

where:

Attribute	Required?	Description
`description`	No	A description of the cell link.
`image`	No	A custom image that points to the defined link. If a link is specified, but not the image, a default information image is displayed. If a custom image is specified but no link is defined, the image will still appear. The URL to the image can be either absolute or relative: • For absolute URLs, the string should begin with "`http://`". • For relative URLs: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. – Starting the string without a slash indicates that the URL is relative to the current document.

Attribute	Required?	Description
image_align	No	Sets the position of the custom image specified by the image attribute. Valid values are: • LEFT: Place image before the cell contents (default) • RIGHT: Place image after the cell contents
link	Yes	A hyperlink for HTML rendering or a call to a JavaScript method. When the link is to a URL, it always opens in a new window. **Note:** The URL can be an absolute or relative URL: • For absolute URLs, the string should begin with "http://". • For relative URLs: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. – Starting the string without a slash indicates that the URL is relative to the current document. The following entities in the link value are replaced by the indicated values when the link is generated. Entities must start with an ampersand (&) and end with a semicolon (;). • &Description; — the cell alert description • &Value; — the cell value • &ColHeader; — ampersand-separated dimension/member value pairs • &RowHeader; —ampersand-separated dimension/member value pairs • &ColIndex; — the display index (0-based) of the column in the grid • &RowIndex; — the display index (0-based) of the row in the grid For an example of how the replacement variables work, see the Examples section for this property.

Attribute	Required?	Description
scope	No	The cells to which the link should be applied, specified as a series of dimension and member sets enclosed in braces. Use unique names to ensure the right member is found. In Essbase, using display names will not work if DataBlox's useAliases is set to false (users can set this through the user interface). For MSAS and Alphablox Cube Server data sources, use unique names starting with the cube name to ensure the correct member at the correct level is found. For SAP BW, always use unique names starting with the query name. Using display names will result in data exception or random data. The scope applies to all axes of the result set, not just the row and column axes. Specify the scope as follows: `scope="{d0:m00[,m01,... m0n]} {d1:m10[,m11,...m1n]}..."` where d0 denotes a dimension and m00 denotes a member within that dimension. For example, for Essbase: `scope="{Product:Coke} {Scenario: Actual, Budget}"` For MSAS, Alphablox Cube Server, and SAP BW: `scope="{[My Cube].[Product]:[Code]}` `{[My Cube].[Scenario]: [Actual], [Budget]}"` For relational data sources, use "{columns: *columnName* [,*columnName2*,...]}". The following member search functions are available for specifying the level of members the link should be applied to: • Leaf(): the leaf-level descendants of the specified member. Only one member can be specified in the function. Example: scope="{Market:leaf(East)}" (Essbase) • Child(): the children of the specified member. Only one member can be specified in the function. Example: scope="{Market:child(East)}" (Essbase) • Descendants(): all descendants of the specified member. Only one member can be specified in the function. Example: scope="{[My Cube].[Market]:descendants([East])}" (MSAS) • Gen(): all members of the specified generation. Example: scope="{Market:gen(2)}" (Essbase) • Not(): members to which the cell link should not be applied. You can specify multiple members, separated by a comma. Example: scope="{[My Cube].[Market]:not([East], [West])}" (MSAS) The function names are case-insensitive. You can combine the functions in the scope statement. See "Scope Examples" on page 228 for more examples.
index	No	Note: This attribute is only valid as a cellLink tag attribute. When using the setCellLink Java method, use the id argument instead. The number of the cell link to define. If you do not specify this attribute, the next available cell link number is used. For example, if cell link 1-4 are already defined, cell link 5 is used.

Usage

The `cellLink` property allows you to specify the conditions under which cells will point to a defined hyperlink for HTML rendering or a call to a JavaScript method. The number of the cell link dictates the order in which it is evaluated, starting with the first cellLink (`index="1"`). The first defined cell link that matches the cell's condition and scope is the only link applied to that cell. Be sure to consider possible overlaps when defining cell links.

- If you do not specify a given property, its default value applies to the cell.
- Links defined using `cellLink` do appear in editable cells defined using cellEditor.
- Links defined using cellAlert take precedence over links defined with `cellLink`. On a given cell, if both a `cellAlert` containing a link and a `cellLink` are defined and the conditions for both parameters are true, the `cellAlert` link is used. If the condition of the `cellAlert` is not true, the `cellLink` is used.
- Unique names (base names in IBM DB2 OLAP Server or Hyperion Essbase) or display names can be used for the dimension and member name string specified in the `scope`. Using unique names allows you to differentiate between different members or dimensions with the same display names. In IBM DB2 OLAP Server or Hyperion Essbase, you can specify a member, regardless of the alias table in use, by using the base name.
- For MSAS and Alphablox Cube Server data sources, when specifying the scope:
  - Enclose dimension names and member names in brackets, starting from the cube name. For example, ″[California]″ or ″[My Cube].[Market].[West].[California]″.
  - If a member name is not enclosed in brackets, it will be treated as a member whose name is all uppercase.

Examples

The following example adds a cell link to all cells in {Market:Central} regardless of the cell values. The cell link indicator is displayed to the left of the cell contents. When users click on the links, the page "www.ibm.com" is displayed in a new browser window.

```
<blox:cellLink
   index="3"
   scope="{[Cube].[Market]:[Central]}"
   description="Cells with the DB2 Alphablox link"
   link="http://www.ibm.com"
   image="myIcon.gif"
   image_align="left" />
```

In the following example, the link for the cell alert is set to:

```
link="decoderequest.jsp?row=&RowHeader;&column=&ColHeader;&value=&Value;
&rowIndex=&RowIndex;&colIndex=&ColIndex;"
```

If a user clicks on the cell for Q3, John Bob in the following Grid:

Time	Hank	Jack	Mary Lou	John Bob
Q1		*(i)***115551.471**	14025.051	82578.896
Q2	13135.487	*(i)***117395.421**	34878.844	39445.495
Q3	*(i)***191617.066**	93620.337	51401.572	*(i)***111110.831**
Q4	24777.37	84440.596	#No Access	44903.466

the URL passed through will be:

```
decoderequest.jsp?row=Time=Q3&column=Customer=John%20Bob&value=111110.831&
rowIndex=2&colIndex=3
```

See Also

"cellAlert" on page 223, "cellEditor" on page 229, "cellFormat" on page 231

columnHeadersWrapped

Specifies whether multi-word headings in grid column headers should be wrapped onto more than one line.

Data Sources

All

Syntax

```
columnHeadersWrapped="wrapped"
```

where:

Argument	Default	Description
wrapped	false	Specify true to enable wrapping of column headers.

Usage

When this tag attribute is set to false (the default), the column width is sized to fit the entire column header text without wrapping. When this property is set to true, the column header text will wrap to reduce column width. The width of the column will be automatically determined to ensure the longest word in the header and the longest data in the data cell will fit.

See Also

"rowHeadersWrapped" on page 253

columnWidths

Specifies the widths of the columns in the grid.

Data Sources

All

Syntax

```
columnWidths="widths"
```

where:

Argument	Default	Description
widths	null	A comma-delimited string of integers that defines column widths in pixels.

Usage

The autosizeEnabled property needs to be false for this property to be used. The browser automatically determines the column widths to fit the longest data value. If you explicitly set the column widths, the value will be ignored if the value to be displayed in a column exceeds the defined width.

See Also
"autosizeEnabled" on page 222

commentsEnabled

Specifies whether 1) the menu items for adding and displaying comments are displayed in the Grid's right-click menu and 2) whether to display the comment indicator on the upper right corner when cell comments are available.

Data Sources
Multidimensional

Syntax
```
commentsEnabled = "boolean"
```

where:

Argument	Default	Description
enabled	false	Whether the Comments menu item and its sub menu items Add Comments and Display Comments are available to users in the Grid cell's right-click menu. When set to false, these choices will not be displayed and the comment indicator (a red triangle) on the cell's upper right corner will not display when there are comments associated with the cell.

See Also
Chapter 8, "CommentsBlox Tag Reference," on page 139.

defaultCellFormat

Specifies the default format mask for all data values in the grid.

Data Sources
All

Syntax
```
defaultCellFormat="mask"
```

where:

Argument	Default	Description
mask	empty string	String that defines cell formatting attributes. For more information on the format string, see the examples below.

Usage
The formatting specified by defaultCellFormat is applied when no other style format exists (either through the cellFormat property or through the user interface).

Note the following about the defaultCellFormat property:

- The cell format displayed in the Blox user interface is dependent on the client's locale. It uses the first language in the browser's language list.
- Do not use the backslash escape character (\) in the format string.

- To display a symbol such as the percent sign (%), use single quotes. If the symbol to display is a double quote, then precede it with a backslash escape character (\).
- In some virtual machines (Sun 1.1.6, for example), the number mask (i.e., `#,###.00`) must be the same for the positive and negative masks. If they are not, the negative mask does not work properly.
- For IBM DB2 OLAP Server or Hyperion Essbase data sources, use `defaultCellFormat` or cellFormat to control the formatting of data values instead of the `{DECIMAL}` report script command.

Examples

- Red negatives: #,###.00;[red]-#,###.00

  ```
  defaultCellFormat="#,###.00;[red]-#,###.00"
  ```
- Parenthesis around red negatives: #,###.00;[red](#,###.00)

  ```
  defaultCellFormat="#,###.00;[red](#,###.00)"
  ```
- Millions: #,###M

  ```
  defaultCellFormat="#,###M"
  ```
- Thousands: #,###K

  ```
  defaultCellFormat="#,###K"
  ```
- Percent with 2 decimal places: ###.00'%'

  ```
  defaultCellFormat="###.00'%'"
  ```
- Show integers padded with zeros to 6 digits: 000000

  ```
  defaultCellFormat="000000"
  ```
- Show two places of decimals (regardless of the precision of the underlying value: #,###.00

  ```
  defaultCellFormat="#,###.00"
  ```

See Also

"cellFormat" on page 231, "formatMask" on page 246

drillThroughEnabled

Specifies if drillthrough operation is enabled on the GridBlox user interface.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services

Syntax

`drillThroughEnabled="`*`drillThroughEnabled`*`"`

Usage

For IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services, this is for data sources which have drillthrough reports set up through IBM DB2 OLAP Server Integration Services or Essbase Integration Services.

When `drillThroughEnabled` is set to `true` on a GridBlox, by default DB2 Alphablox sends the coordinates of the cell where the drillthrough operation is initiated to a RDBResultSetDataBlox and renders the relational detail using a ReportBlox. The report is displayed in a separate, resizable browser window. If the user right-clicks on a different cell and selects Drill through from the right-click menu, another browser window will pop up displaying the relational detail. This allows users to compare detailed data for different cells.

If the cell where the drillthrough operation is triggered is at the lowest level of its hierarchy, only one row set is returned. Otherwise, all of the row sets that make up the source data of that cell are returned. For MSAS, the maximum number of rows that can be returned is determined by the Maximum DrillThrough Rows setting specified in the DB2 Alphablox Home Page's Administration tab, Data Sources link. For IBM DB2 OLAP Server or Hyperion Essbase, this is set in EIS by the Essbase administrator.

To define your own window properties for displaying relational details or to bring up your custom JSP, use the `drillThroughWindow` tag. For details on how this works, see the Drillthrough Support for Microsoft Analysis Services section in the *Developer's Guide*. A live example is available in the Retrieving Data section in Blox Sampler.

See Also

"drillThroughWindow" on page 242. For detail on RDBResultSetDataBlox and ReportBlox, see the *Relational Reporting Developer's Guide*.

drillThroughWindow

Specifies the properties of the popped up browser window when a drillthrough operation is triggered on the GridBlox user interface.

Data Sources

IBM DB2 OLAP Server, Hyperion Essbase, Microsoft Analysis Services

Syntax

```
drillThroughWindow="drillThroughWindowProperties"
```

or

```
<blox:drillThroughWindow
    url=""
    name=""
    height=""
    width=""
    resizable=""
    locationbarVisible=""
    menubarVisible=""
    scrollbarsVisible=""
    statusbarVisible=""
    toolbarVisible=""
>
</blox:drillThroughWindow>
```

where:

Argument	Description
`drillThroughWindowProperties`	A string containing a comma-delimited name-value pair that specifies the window properties. Valid names in the string are the following tag attributes for the `drillThroughWindow` tag: • `url`: a String containing the URL of the JSP page to load into the popped-up window • `name`: a String containing the name of the popped-up window • `height`: the height (in pixels) of the popped-up window

- `width`: the width (in pixels) of the popped-up window
- `resizable`: `true` or `false`; whether the popped-up window is resizable. The default is `true`.
- `locationbarVisible`: `true` or `false`; whether the location bar should be visible in the popped-up window. The default is `true`.
- `menubarVisible`: `true` or `false`; whether the menu bar should be visible in the popped-up window. The default is `true`.
- `scrollbarsVisible`: `true` or `false`; whether the scroll bars should be visible in the popped-up window. The default is `true`.
- `statusbarVisible`: `true` or `false`; whether the status bar should be visible in the popped-up window. The default is `true`.
- `toolbarVisible`: `true` or `false`; whether the toolbar (the browser's toolbar) should be visible in the popped up window. The default is `true`.

Usage

For IBM DB2 OLAP Server, IBM DB2 OLAP Server Deployment Services, Hyperion Essbase, or Essbase Deployment Services, this is for data sources which have drillthrough reports set up through IBM DB2 OLAP Server Integration Services or Essbase Integration Services.

When `drillThroughEnabled` is set to `true` on a GridBlox, by default DB2 Alphablox sends the coordinates of the cell where the drillthrough operation is initiated to a RDBResultSetDataBlox and renders the relational detail using a ReportBlox. The report will be displayed in a popped-up browser window. This popped up browser window, by default, is resizable, with its toolbar, scroll bars, menu bar, status bar, and location bar visible.

If you want to specify your own window properties for the popped-up browser window, specify a comma delimited name-value pair string representing the URL, name, and/or features of the drillthrough window. The window properties are similar to those available to JavaScript's window object.

The URL you specify can be of one of the following formats:

- For absolute URLs, the string should begin with "`http://`".
- For relative URLs:
  - Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL.
  - Starting the string without a slash indicates that the URL is relative to the current document.

Examples

```
drillThroughWindow =
"url=myDrillThroughPage.jsp,name=myDrillThroughWindowName,height=600,
width=800,statusbarVisible=false, locationbarVisible=false"
```

See Also

"drillThroughEnabled" on page 241

editableCellStyle

Specifies the foreground and background colors of editable cells.

Data Sources

All, except SAP BW

Syntax

```
editableCellStyle="style"
```

or

```
<blox:editableCellStyle
    background=""
    font=""
    foreground="" >
</blox:editableCellStyle>
```

where:

Argument	Default	Description
`style`	`background=white, foreground=blue`	A comma-delimited string specifying: `foreground`: the cell's text color `background`: the cell's background color `font`: the *font name:style:point* to use Use the color's name or hexadecimal value.

Usage

For font, you can specify the font name, style to use, and the point size using the following syntax:

font name:style:point

- *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace.
- *style*: Valid font styles are: `plain`, `italic`, `bold`, and `bolditalic`
- *point*: An integer for point size (usually 8-36).

If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. The following examples show how to specify the font using the JSP tags:

```
font="Arial:bolditalic:12"
font=":Bold:12"
```

Examples

```
editableCellStyle="background=red, foreground=green, font=Arial:bold:12"
```

See Also

"editedCellStyle" on page 245, "cellEditor" on page 229

editedCellStyle

Specifies the foreground and background colors of cells that have been edited.

Data Sources

All, except SAP BW

Syntax

```
editedCellStyle="style"
```

or

```
<blox:editedCellStyle
    background=""
    font=""
    foreground="">
</blox:editedCellStyle>
```

where:

Argument	Default	Description
`style`	`background=white, foreground=blue`	A comma-delimited string specifying: `foreground`: the cell's text color `background`: the cell's background color `font`: the `font name:style:point` to use Use the color's name or hexadecimal value.

Usage

This property specifies the colors for editable cells after the user has changed a value. Specifying a different color for changed cells provides visual cues for a user who is editing many cells.

For font, you can specify the font name, style to use, and the point size using the following syntax:

`font name:style:point`

- *font name*: Acceptable font name values vary widely by browser and client machine. The following font names are generally accepted: Arial, Courier, Helvetica, TimesRoman, SansSerif, Serif, Monospace.
- *style*: Valid font styles are: `plain`, `italic`, `bold`, and `bolditalic`
- *point*: An integer for point size (usually 8-36).

If any of the three attributes is not specified, the default or the currently inherited font value is applied. However, the colons separating the attributes should be included. The following examples show how to specify the font using the JSP tags:

```
font="Arial:bolditalic:12"
font=":Bold:12"
```

Examples

```
editedCellStyle="background=gray, foreground=orange,
font=Helvetica:plain:12"
```

See Also

"editableCellStyle" on page 244, "cellEditor" on page 229

enablePoppedOut

This is a common Blox property. If the GridBlox is nested within a PresentBlox:

- If the `poppedOut` property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the `poppedOut` property and its related properties have not been specified in the PresentBlox, the popped out settings in the nested GridBlox are applied to the PresentBlox.

For a complete description, see "enablePoppedOut" on page 152.

expandCollapseMode

Specifies whether the grid should display the expand and collapse (plus and minus) signs on members.

Data Sources

All

Syntax

```
expandCollapseMode="expandCollapseMode"
```

where:

Argument	Default	Description
`expandCollapseMode`	`false`	Set to `true` to enable expand and collapse; `false` to disable it.

Usage

When `expandCollapseMode` is set to `true`, plus and minus signs are displayed to indicate a drill up or drill down operation in the user interface of GridBlox. Note that if you want parent members to come first before their children, you should set DataBlox `parentFirst` property to `true` rather than do so through the query. This is to ensure the expand/collapse mode can search through the result set correctly to determine the base members and shared members.

Examples

```
expandCollapseMode="true"
```

formatMask

Specifies a predefined format mask for cells when using the format mask user interface.

Data Sources

All

Syntax

```
<blox:formatMask
       index="maskNumber"
       mask="mask"
/>
```

where:

Argument	Default	Description
maskNumber	null	The index number of the mask to define or retrieve. Must be an integer between 1 and 15.
mask	empty string	String that defines formatting attributes.

Usage

Unlike defaultCellFormat or cellFormat, this property has no effect on the grid itself, it only effects what appears in the Apply Format Mask dialog.

The following table lists the predefined mask values, and their associated format names, for each mask. The format names and masks may be different for language versions other than English. You can create your own number masks, beginning with 12, in addition to the predefined ones.

Format Mask number	Mask	Format Name
formatMask1		No mask
formatMask2	#,##0.00;[red]-#,##0.00	Negative red
formatMask3	#,##0.00;[red](#,##0.00)	Negative red parenthesis
formatMask4	#,##0.00;(#,##0.00)	Parenthesis
formatMask5	#,###K	Thousands
formatMask6	#,###M	Millions
formatMask7	##0.00'%'	Percentage
formatMask8	$,##0	Dollars
formatMask9	#,##0.00;(#,##0.00)	Euros
formatMask10	#,##0.00	2 decimal places
formatMask11	0	Integer

Note the following:

- Do not use the backslash escape character (\) in the value string.
- You can use a slash (/) character to divide the one value by another ($#,###/1000, for example).
- This property is used in conjunction with the formatName1-15 property. See the example below and “formatName” on page 248

Examples

The following code would allow the user to select a format name called “Negative red” with an associated number mask of #,##0.00;[red]-#,##0.00 from the format mask user interface.

```
<blox:formatMask
               index="2"
                mask="#,##0.00;[red]-#,##0.00" />
<blox:formatName
               index="2"
                name="Negative red" />
```

See Also

“formatName” on page 248

formatName

Specifies a predefined format name for cells when using the format mask user interface.

Data Sources

All

Syntax

```
<blox:formatName
       index="formatNumber"
       name="name"
/>
```

where:

Argument	Default	Description
formatNumber	null	The index number of the format name to define or retrieve. Must be an integer between 1 and 15.
name	empty string	String defining the name of the specified format mask

Usage

Every predefined formatMask property is assigned a predefined formatName. The property name is one of formatName1 through formatName15.

The following table lists the format name properties, their predefined names, and their associated format masks. The format names and number mask syntax may be different for language versions other than English.

Format Name property	Format Name	Format Mask
formatName1	No mask	
formatName2	Negative red	#,##0.00;[red]-#,##0.00
formatName3	Negative red parenthesis	#,##0.00;[red](#,##0.00)
formatName4	Parenthesis	#,##0.00;(#,##0.00)
formatName5	Thousands	#,###K
formatName6	Millions	#,###M
formatName7	Percentage	##0.00'%'
formatName8	Dollars	$,##0
formatName9	Euros	#,##0.00;(#,##0.00)
formatName10	2 decimal places	#,##0.00
formatName11	Integer	0

Examples

The following code allow the user to select a format name called "Negative red" with an associated number mask of #,##0.00;[red]-#,##0.00 from the format mask user interface.

```
<blox:formatMask
               index="2"
                mask="#,##0.00;[red]-#,##0.00" />
<blox:formatName
               index="2"
                name="Negative red" />
```

See Also

"formatMask" on page 246

gridLinesVisible

Specifies whether lines appear between each cell in the grid.

Data Sources

All

Syntax

```
gridLinesVisible="visible"
```

where:

Argument	Default	Description
visible	true	Specify true to display grid lines; false to hide them.

Examples

```
gridLinesVisible="false"
```

headingIconsVisible

Specifies heading icons for header links appear on row and column headings.

Data Sources

All

Syntax

```
headingIconsVisible="visible"
```

where:

Argument	Default	Description
visible	true	Specify true to display heading icons; false to hide them.

Examples

```
headingIconsVisible="false"
```

headingsEnabled

Specifies whether row and column headings appear when the grid is printed.

Data Sources

All

Syntax

```
headingsEnabled="enable"
```

where:

Argument	Default	Description
enable	true	Specify true to display headings; false to hide them.

Examples

```
headingsEnabled="false"
```

height

This is a common Blox property. For a complete description, see “height” on page 35.

helpTargetFrame

This is a common Blox property. For a complete description, see “helpTargetFrame” on page 35.

htmlGridScrolling

Specifies whether to display scroll bars on the grid.

Data Sources

All

Syntax

```
htmlGridScrolling="scroll"
```

where:

Argument	Default	Description
`scroll`	`true`	Specify `true` to display scroll bars when necessary; `false` not to display them.

Usage

Setting this property to `true` causes the scroll bars to appear only when needed. If the display area can accommodate all the requested data or if the value is `false`, no scroll bars appear.

htmlShowFullTable

Specifies if all rows and columns in the grid to appear (ignoring the defined Blox area and the setting of the htmlGridScrolling property). Scrollbars are not part of the display. If the contents of the grid require it, the data may extend beyond the viewable area on the screen. In this case, HTML page scroll bars enable the user to scroll to and view off-screen data. The default is `false`, which causes the display to stay within the Blox bounds and not display the full table.

Data Sources

All

Syntax

```
htmlShowFullTable="show"
```

where:

Argument	Default	Description
`show`	`false`	Specify `true` to display all rows and columns on the grid; `false` to use scroll bars if they do not fit in the space.

informationWindowName

Specifies the name of the window used for displaying HTML pages defined in the Header Links for a particular application.

Data Sources

All

Syntax

```
informationWindowName="name"
```

where:

Argument	Default	Description
name	"Information"	String representing the window name.

Usage

By defining a window name using this property, all header link URLs will be opened in the defined window, rather than opening new windows for each URL.

maximumUndoSteps

This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 36.

menubarVisible

This is a common Blox property. For a complete description, see "menubarVisible" on page 37.

missingValueString

Specifies a string to display in a cell for which there is no data in the database.

Data Sources

All

Syntax

```
missingValueString="value"
```

where:

Argument	Default	Description
value	empty string	Any string

Usage

When accessing relational data sources, the message appears when a cell has a null value.

Examples

```
missingValueString="Data is missing"
```

See Also

"noAccessValueString" on page 252

noAccessValueString

Specifies the string to display in a grid cell for which the user has not been granted data access.

Data Sources

Multidimensional

Syntax

```
noAccessValueString="value"
```

where:

Argument	Default	Description
value	#No Access	Any string

Examples

```
noAccessValueString="Access denied"
```

See Also

missingValueString

noDataMessage

This is a common Blox property. For a complete description, see "noDataMessage" on page 37.

poppedOut

This is a property inherited from ContainerBlox. If the GridBlox is nested within a PresentBlox:

- If the `poppedOut` property and its related properties have been specified in the PresentBlox, the settings in the PresentBlox are used.
- If the `poppedOut` property and its related properties have not been specified in the PresentBlox, the popped out settings in the nested GridBlox are applied to the PresentBlox.

For a complete description, see "poppedOut" on page 153.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 154.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 154.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 155.

relationalRowNumbersOn

Specifies whether to display the row numbers from a relational data source.

Data Sources
Relational

Syntax
```
relationalRowNumbersOn="enable"
```

where:

Argument	Default	Description
enable	false	Specify true to display row numbers; false to hide them.

Examples
```
relationalRowNumbersOn="true"
```

See Also
"rowHeadingsVisible" on page 254

removeAction
This is a common Blox property. For a complete description, see "removeAction" on page 38.

render
This is a common Blox property. For a complete description, see "render" on page 38.

rightClickMenuEnabled
This is a common Blox property. For a complete description, see "rightClickMenuEnabled" on page 39.

rowHeadersWrapped
Specifies whether multi-word headings in grid row headers should be wrapped onto more than one line.

Data Sources
All

Syntax
```
rowHeadersWrapped="wrapped"
```

where:

Argument	Default	Description
wrapped	false	Specify true to enable wrapping of row headers.

Usage
When this property is set to false (the default), the row width is sized to fit the entire row header text without wrapping. When this property is set to true, the row header text will wrap to reduce the width of row header column.

See Also
"columnHeadersWrapped" on page 239

rowHeadingsVisible

Specifies whether the row dimension names above the row headings should appear on the grid.

Data Sources

Multidimensional

Syntax

```
rowHeadingsVisible="visible"
```

where:

Argument	Default	Description
visible	true	Specify true to display row dimension names; false to hide them.

Examples

```
rowHeadingsVisible="false"
```

See Also

"relationalRowNumbersOn" on page 252

rowHeadingWidths

Specifies the widths of the row headings on the grid.

Data Sources

All

Syntax

```
rowHeadingWidths="widths"
```

where:

Argument	Default	Description
widths	none	A comma-separated list of integers, each representing the row headers in pixels.

Usage

The autosizeEnabled property needs to be false for this property to be used. If the width for a row heading is not specified, the width will be automatically calculated to fit the entire heading. If the heading is longer than the width specified, the grid may not display properly.

See Also

"autosizeEnabled" on page 222

rowHeight

Specifies the height (in pixels) for each row.

Data Sources

All

Syntax
`rowHeight="height"`

where:

Argument	Default	Description
`height`	`-1`	An integer representing the row height in pixels

Usage
The autosizeEnabled property needs to be set to `false`. The default (`-1`) sets the row height to an appropriate value for the selected font.

Examples
```
rowHeight="15"
```

See Also
"autosizeEnabled" on page 222

rowIndentation
Specifies whether (and how) to indent row headings.

Data Sources
Multidimensional

Syntax
`rowIndentation="strType"`

where:

Argument	Default	Description
`strType`	`parentLeft`	Possible values (case-sensitive): `parentRight`: The lowest-generation child appears farthest to the left, with each parent indented slightly to the right. `parentLeft`: The lowest-generation child appears farthest to the right, with each parent slightly to the left. `none`: No indentation.

Usage
Indenting row headings helps to indicate the dimension hierarchy.

Examples
```
rowIndentation="none"
```

See Also
"rowHeadingsVisible" on page 254

showColumnDataGeneration
Enables the use of generation styles for columns.

Data Sources
Multidimensional

Syntax

```
showColumnDataGeneration="show"
```

where:

Argument	Default	Description
show	false	Specify true to use generation styles; false not to use them.

Usage

The showColumnDataGeneration and showRowDataGeneration properties must be set to true to apply generation styles to the data cells.

The styles are all set from the theme currently in use. Therefore, you should set the styles in the theme to control row data generation styles (csClmnDtGnrtn0, csClmnDtGnrtn1,... csClmnDtGnrtn*N* classes). The style sheet for the supported themes are in <alphablox>/repository/theme/{*themeName*}. Also, any column styles will override row styles in cells where the rows and columns intersect.

Examples

```
showColumnDataGeneration="true"
```

See Also

"showRowDataGeneration" on page 256

showColumnHeaderGeneration

Enables the use of generation styles for column headers.

Data Sources

Multidimensional

Syntax

```
showColumnHeaderGeneration="show"
```

where:

Argument	Default	Description
show	false	Specify true to use generation styles; false not to use them.

Usage

When the value is set to true, a predefined style for each data generation is applied to the header text. To customize the style, modify the csClmnHdrGnrtn0, csClmnHdrGnrtn1,... csClmnHdrGnrtn*N* classes in the underlying theme. The stylesheet for the supported themes are in <alphablox>/repository/theme/{*themeName*}.

See Also

"showRowHeaderGeneration" on page 257

showRowDataGeneration

Enables the use of generation styles for rows.

Data Sources
Multidimensional

Syntax
`showRowDataGeneration="show"`

where:

Argument	Default	Description
`show`	`false`	Specify `true` to use generation styles; `false` not to use them.

Usage
The `showRowDataGeneration` and `showColumnDataGeneration` properties must be set to `true` to apply generation styles to the data cells.

The styles are all set from the theme currently in use. Therefore, you should set the styles in the underlying theme to control row data generation styles (`csRwDtGnrtn0`, `csRwDtGnrtn1`,... `csRwDtGnrtn`*N* style classes). The stylesheet for the supported themes are in `<alphablox>/repository/theme/`{*themeName*}. Also, any column styles will override row styles in cells where the rows and columns intersect.

Examples
```
showRowDataGeneration="true"
```

See Also
"showColumnDataGeneration" on page 255

showRowHeaderGeneration
Enables the use of generation styles for row headers.

Data Sources
Multidimensional

Syntax
`showRowHeaderGeneration=show`

where:

Argument	Default	Description
`show`	`false`	Specify `true` to use generation styles; `false` not to use them.

Usage
When the value is set to `true`, a predefined style for each data generation is applied to the header text. To customize the style, in the DHTML client, modify the `csRwHdrGnrtn0`, `csRwHdrGnrtn1`,... `csRwHdrGnrtn`*N* classes in the underlying theme. The stylesheet for the supported themes are in `<alphablox>/repository/theme/`{*themeName*}.

See Also
"showColumnHeaderGeneration" on page 256

toolbarVisible

Specifies if the toolbar is visible.

Data Sources

All

Syntax

JSP Tag Attribute

```
toolbarVisible="visible"
```

where:

Argument	Default	Description
visible	true	true: the toolbar is visible; false: the toolbar is not visible.

Usage

By default, the toolbar is visible in a standalone GridBlox. If a nested <blox:toolbar> tag is added, its setting overwrites the value of this attribute. For example, the following code will result in a visible toolbar.

```
<blox:grid id="myGrid" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:grid>
```

Tip: toolbarVisible is only a tag attribute, not a GridBlox property, and therefore there are no associated getter or setter methods.

visible

This is a common Blox property. For a complete description, see “visible” on page 40.

width

This is a common Blox property. For a complete description, see “width” on page 40.

writebackEnabled

Permits users to edit cells in the grid.

Data Sources

All, except SAP BW

Syntax

```
writebackEnabled="enabled"
```

where:

Argument	Default	Description
enable	false	Specify true to enable writeback; false to disable it.

Usage

This property must be specified on GridBlox for the associated grid writeback properties and methods to take effect.

Examples

```
writebackEnabled="true"
```

See Also

"cellEditor" on page 229

Chapter 13. MemberFilterBlox Tag Reference

This chapter contains reference material for MemberFilterBlox tag attributes. For general reference information about Blox, see Chapter 3, “General Blox Reference Information,” on page 15. For information on how to use this reference, see Chapter 1, “Using This Reference,” on page 1.

MemberFilterBlox Overview

MemberFilterBlox allows you to present the Member Filter dialog for users to select members. Member Filter is built into the Blox user interface, available to users when they:

- select Member Filter... from a GridBlox’s right-click menu,
- select Member Filter... from the drop down lists in the Data Layout panel, or
- select More... from the Page panel.

With MemberFilterBlox, you can specify a DataBlox to use and then put a Member Filter dialog on the page to allow your users to select members from all available dimensions or just the dimensions you specified. This dimension selection drop list is populated based on the data query for the underlying DataBlox.

If the underlying DataBlox is used in a PresentBlox on the same page, the PresentBlox will automatically reflect the selections made.

By default, the `dimensionSelectionEnabled` property is set to `true`, making all available dimensions as a result of the data query appear in the list. These dimensions are listed in alphabetical order. Unless specified otherwise, the first dimension in the list is the initial selected dimension and it will appear in the Dimension Hierarchy panel on the left. To specify the initial selected dimension, use the selectedDimension property. To limit the dimensions you want to appear in the drop list, you can specify the list of dimensions using the selectableDimensions property.

MemberFilterBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a MemberFilterBlox. For a copy and paste version of the tag with all the attributes, see “MemberFilterBlox JSP Custom Tag” on page 464.

Syntax

```
<blox:memberFilter
    [attribute="value"] >
    <blox:data bloxRef="" />
</blox:memberFilter>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
applyButtonEnabled
bloxEnabled
bloxName
dimensionSelectionEnabled
height
selectableDimensions
selectedDimension
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:memberFilter>` tag with a closing slash (/) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is `width`, the end of the tag looks as follows:

```
selectedDimension="All Products" />
```

Examples

```
<blox:data id="myDataBlox"
    dataSourceName="QCC-Essbase"
    query="!" />

<blox:memberFilter id="myMemberFilter">
    <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
```

MemberFilterBlox Examples

This section provides examples that demonstrate how MemberFilterBlox can be used as a utility to filter members for all available dimensions in a PresentBlox, to filter members for only the specified dimensions, or to filter only one dimension.

Example 1: Filtering Members for All Available Dimensions

This example adds a MemberFilterBlox on the same page as the PresentBlox using the same DataBlox.

1. A MemberFilterBlox is added to the page using the `<blox:memberFilter>` tag.
2. The dimension selection drop list is enabled. Since no `selectableDimensions` are specified, all dimensions available from the DataBlox will appear in the drop list.
3. The initial selected dimension in the drop list is set to All Time Periods.

4. Use the bloxRef tag attribute to specify the underlying DataBlox.
5. The same DataBlox is used in a PresentBlox. The selections the users make in MemberFilterBlox will be automatically reflected in the PresentBlox.

```
      <%@ taglib uri="bloxtld" prefix="blox"%>

      <blox:data id="myDataBlox"
          dataSourceName="QCC-Essbase"
          useAliases="true"
          query="<ROW (\"All Products\") <ICHILD \"All Products\"
                 <COLUMN (\"All Time Periods\" Measures Scenario)
                 <CHILD \"All Time Periods\" !"
          selectableSlicerDimensions="All Locations" />

      <html>
      <head>
          <blox:header />
      </head>
      <body>
(1)   <blox:memberFilter id="memberFilterBlox"
(2)       dimensionSelectionEnabled = "true"
(3)       selectedDimension="All Time Periods">
(4)       <blox:data bloxRef="myDataBlox" />
      </blox:memberFilter>
      <br>
      <blox:present id="myPresentBlox" width="600" height="400">
(5)       <blox:data bloxRef="myDataBlox" />
      </blox:present>
      </body>
      </html>
```

Example 2: Filtering Members for Specified Dimensions Only

This example adds a MemberFilterBlox that only has the All Products and All Time Periods dimensions on the dimension selection drop list using the selectableDimensions tag attribute.

```
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:data id="myDataBlox"
   dataSourceName="QCC-Essbase"
   useAliases="true"
   query="<ROW (\"All Products\") <ICHILD \"All Products\"
      <COLUMN (\"All Time Periods\" Measures Scenario)
      <CHILD \"All Time Periods\" !" />
...

<blox:memberFilter id="memberFilterBlox"
   dimensionSelectionEnabled="true"
   selectableDimensions="All Products, All Time Periods">
   <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
...
```

Example 3: Filtering Members for One Dimension Only

This examples adds a MemberFilterBlox that does not have a dimension selection drop list (dimensionSelectionEnabled = "false"). With selectedDimension set to All Products, All Products will show up in the left Dimension Hierarchy panel. All the users can do is to select members from this dimension.

```
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:data id="myDataBlox"
   dataSourceName="QCC-Essbase"
```

```
   useAliases="true"
   query="<ROW (\"All Products\") <ICHILD \"All Products\"
      <COLUMN (\"All Time Periods\" Measures Scenario)
      <CHILD \"All Time Periods\" !" />
...

<blox:memberFilter id="memberFilterLocked"
   dimensionSelectionEnabled="false"
   selectedDimension="All Products">
   <blox:data bloxRef="myDataBlox" />
</blox:memberFilter>
...
```

Unique MemberFilterBlox Tag Attributes

The following are unique MemberFilterBlox tag attributes. For lists of tag attributes common to several Blox, see "Common Blox Tag Attributes by Category" on page 29.

- `applyButtonEnabled`
- `dimensionSelectionEnabled`
- `selectableDimensions`
- `selectedDimension`

MemberFilterBlox Tag Attributes

This section describes the tag attributes supported by MemberFilterBlox in alphabetical order. Common Blox tag attributes available are listed but not described. For complete descriptions of common Blox properties, see "Tag Attributes Common to Multiple Blox" on page 30. For reference information on MemberFilterBlox methods, see the MemberFilterBlox class in the com.alphablox.blox package in the Javadoc documentation.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 36.

applyButtonEnabled

Shows the Apply button in the Member Filter user interface.

Data Sources

Multidimensional

Syntax

`applyButtonEnabled = "`*`applyButtonEnabled`*`"`

where:

Argument	Default	Description
`applyButtonEnabled`	`true`	`true`: shows the Apply button; `false`: hides the Apply button.

Usage

Sometimes you may not need the Apply button in the Member Filter because the DataBlox referenced in the MemberFilterBlox is not used in a user interface Blox

on the same page, and the MemberFilterBlox is only used to let users specify the members of interest in order to construct a separate query or perform some calculation. To hide the Apply button, set this property to `false`.

bloxEnabled

This is a common Blox property. For a complete description, see "bloxEnabled" on page 32.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 32.

dimensionSelectionEnabled

Specifies whether the dimension selection drop list should be displayed.

Data Sources

Multidimensional

Syntax

```
dimensionSelectionEnabled = "dimensionSelectionEnabled"
```

where:

Argument	Default	Description
`dimensionSelectionEnabled`	`true`	`true`: display the dimension selection drop list; `false`: hide the dimension selection drop list.

Usage

When `dimensionSelectionEnabled` is set to `true`, all available dimensions as a result of the data query appear in the drop list. The dimensions are listed in alphabetical order. By default, the first dimension in the list is the initial selected dimension and appears in the Dimension Hierarchy panel on the left. To specify a different dimension as the initial selection, use selectedDimension. To limit the dimensions in the drop list, use selectableDimensions.

See Also

"selectedDimension" on page 266, "selectableDimensions" on page 265

height

This is a common Blox property. For a complete description, see "height" on page 35.

Member Filter has a default width and height for best layout. Do not specify the width or height unless you really need a specific size. If the size you specify is too small (less than 325 pixels in height and 600 pixels in width), the size will be set to the 325 X 600.

selectableDimensions

Specifies the dimensions to appear in the dimension selection drop list.

Data Sources
Multidimensional

Syntax
```
selectableDimensions = "selectableDimensions"
```

where:

Argument	Default	Description
selectableDimensions	All available dimensions in the result set	A comma-separated list of dimensions.

Usage
When dimensionSelectionEnabled is set to true (the default), all available dimensions in the data result set appear in the dimension selection drop list unless specified otherwise in selectableDimensions. The dimensions you specified always appear in alphabetical order in the drop list regardless of the order you specified them. The initial selected dimension (the dimension that appears in the Dimension Hierarchy panel) is the first one on the list unless specified otherwise using selectedDimension.

Examples
```
<blox:memberFilter id="myMemberFilter"
   dimensionSelectionEnabled = "true"
   selectableDimensions = "Year, Scenario, Products">
   <blox:data bloxRef = "myDataBlox" />
</blox:memberFilter>
```

See Also
"dimensionSelectionEnabled" on page 265, "selectedDimension" on page 266

selectedDimension
Specifies the initial selected dimension.

Data Sources
Multidimensional

Syntax
```
seletedDimension = "selectedDimension"
```

where:

Argument	Default	Description
selectedDimension	The first dimension among the available dimensions from the data query	The name of the dimension to be the initial selected dimension.

Usage
Since only members from one dimension can be displayed in the Dimension Hierarchy panel, you should specify the initial selected dimension. If this is not specified, the first dimension in alphabetical order will be the selected dimension.

visible

This is a common Blox property. For a complete description, see “visible” on page 40.

width

This is a common Blox property. For a complete description, see “width” on page 40

Member Filter has a default width and height for best layout. Do not specify the width or height unless you really need a specific size. If the size you specify is too small (less than 325 pixels in height and 600 pixels in width), the size will be set to the 325 X 600.

Chapter 14. PageBlox Tag Reference

This chapter contains reference material for PageBlox tag attributes. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

PageBlox Overview

PageBlox enables users to filter data that appears in the grid or chart. Each dimension in the current result set that resides on the Page axis appears as a drop list in the Page Filter. When the user selects a dimension member from the drop list, the member is used to filter the data appearing in the grid or chart.

PageBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a PageBlox. For a copy and paste version of the tag with all the attributes, see "PageBlox JSP Custom Tag" on page 465.

Syntax

```
<blox:page
    [attribute="value"] >
</blox:page>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
bookmarkFilter
fixedChoiceLists
height
helpTargetFrame
labelPlacement
maximumUndoSteps
moreChoicesEnabled

Attribute
moreChoicesEnabledDefault
noDataMessage
render
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:page>` tag with a closing slash (/) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is `width`, the end of the tag looks as follows:

```
width="650" />
```

Examples

```
<blox:page
     fixedChoiceLists="Year:Qtr1,Qtr2;Market:East"
     >
</blox:page>
```

PageBlox Tag Attributes by Category

The following tables list unique PageBlox tag attributes. For lists of tag attributes common to several Blox, see “Common Blox Tag Attributes by Category” on page 29. For reference information on PageBlox methods, see the PageBlox class in the com.alphablox.blox package in the Javadoc documentation.

The tag attributes supported by PageBlox are organized in the cross reference as follows:

Choice Lists

The following are tag attributes associated with the choice lists for PageBlox:

- `fixedChoiceLists`
- `moreChoicesEnabledDefault`
- `moreChoicesEnabledDefault`

Panel Type and Appearance

The following are tag attributes for setting the page panel type and for settings that affect the PageBlox appearance:

- `labelPlacement`

PageBlox Tag Attributes

This section describes the tag attributes supported by PageBlox in alphabetical order. Common Blox tag attributes are listed but not described. For complete descriptions of common Blox tag attributes, see "Tag Attributes Common to Multiple Blox" on page 30. For reference information on PageBlox methods, see the PageBlox class in the com.alphablox.blox package in the Javadoc documentation.

id

This is a common Blox property. For a complete description, see "id" on page 36.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see "applyPropertiesAfterBookmark" on page 30.

bloxEnabled

This is a common Blox property. For a complete description, see "bloxEnabled" on page 32.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 32.

bookmarkFilter

This is a common Blox property. For a complete description, see "bookmarkFilter" on page 31.

fixedChoiceLists

Places the named dimensions and members on the drop list so that the user can access them.

Data Sources

Multidimensional

Syntax

`fixedChoiceLists="`*`dimensionMemberList`*`"`

where:

Argument	Default	Description
`dimensionMemberList`	empty string	Pairs of comma-delimited string of member names of the specified dimension, separated by semicolons: *dimension1:member1,member2; dimension2:member1,member2* There should be no spaces in between unless the dimension or member names contain one.

Usage

The default permits the user to access all dimensions and members.

Any dimension(s) specified in the `fixedChoiceLists` property must also appear in the DataBlox selectableSlicerDimensions property.

The query for the initial display must include only one of the members for each dimension specified in the `fixedChoiseLists` property, otherwise the root level member of the dimension initially appears in the fixed choice list. The member(s) specified in the query will be the default member(s) in the fixed choice list(s). For example, if you have a fixed choice list on the `Year` dimension with members `Q1` and `Q2` and neither `Q1` or `Q2` is specified in the query, then the fixed choice list shows `Year`, `Q1`, and `Q2` initially. After selecting `Q1` or `Q2` in the page filter, `Year` is then removed from the list. If two or more members are specified in the query, the fixed choice list will not appear in the PageBlox.

If the PageBlox is nested within another Blox (for example, a PresentBlox) and a user moves a member from the PageBlox to a row or column axis (for example), then the fixed choice list does not apply to that row and column axis; it only applies to the PageBlox. If you do not want this to occur, use a standalone PageBlox instead of a nested PageBlox.

For Microsoft Analysis Services and DB2 Alphablox Cube Server, use unique names for better performance. For IBM DB2 OLAP Server or Hyperion Essbase, you can use unique names (base names) or display names. This allows you to differentiate between different members or dimensions with the same display names.

In IBM DB2 OLAP Server or Hyperion Essbase, you can specify a member, regardless of the alias table in use, by using the base name.

Examples

```
fixedChoiceLists="Year:Qtr1,Qtr2;Market:East"
```

See Also

"moreChoicesEnabled" on page 273, "moreChoicesEnabledDefault" on page 273, "selectableSlicerDimensions" on page 202

height

This is a common Blox property. For a complete description, see "height" on page 35.

helpTargetFrame

This is a common Blox property. For a complete description, see "helpTargetFrame" on page 35.

labelPlacement

Sets the placement of the PageBlox label relative to the PageBlox drop down list. Valid `labelPlacement` values are `left`, `top`, and `none`. The default value is `left`.

Data Sources

Multidimensional

Syntax

```
labelPlacement="placement"
```

where:

Argument	Default	Description
`placement`	`left`	A String indicating where the PageBlox label is placed relative to the drop down list. Valid values are `left`, `top`, and `none`.

Examples

```
labelPlacement="top"
```

maximumUndoSteps

This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 36.

moreChoicesEnabled

Specifies whether the "More..." option in the PageBlox drop list for the named dimensions should be available to permit the user to view more choices using the Member Filter.

Data Sources

Multidimensional

Syntax

```
moreChoicesEnabled="choices"
```

where:

Argument	Default	Description
`choices`	empty string	A string of *dimension:enabled* pairs separated by a semicolon (";"). The list should not contain any spaces, unless it is a space in the dimension name. • *dimension*: dimension name • *enabled*: `true` or `false`

Usage

By default, the **More...** option is available for all dimensions placed on the page axis. This property allows you to hide the **More...** option for the named dimensions so the user cannot access Member Filter for more choices.

Examples

The following lines make the More... option available in the PageBlox drop lists for the Product dimension but not for the Market dimension. Note that there is no space in between unless the dimension name contains one.

```
moreChoicesEnabled="Product:true;Market:false"
```

See Also

"fixedChoiceLists" on page 271, "moreChoicesEnabledDefault" on page 273

moreChoicesEnabledDefault

Specifies the default for whether the "More..." option in the PageBlox drop list for the named dimensions should be available to permit the user to view more choices using the Member Filter.

Data Sources
Multidimensional

Syntax
```
moreChoicesEnabledDefault="boolean"
```

where:

Argument	Default	Description
enabledDefault	true	Valid values are true and false.

Examples
```
moreChoicesEnabledDefault="false"
```

See Also
"fixedChoiceLists" on page 271, "moreChoicesEnabled" on page 273

noDataMessage
This is a common Blox property. For a complete description, see "noDataMessage" on page 37.

render
This is a common Blox property. For a complete description, see "render" on page 38.

visible
This is a common Blox property. For a complete description, see "visible" on page 40.

width
This is a common Blox property. For a complete description, see "width" on page 40.

Chapter 15. PresentBlox Tag Reference

This chapter contains reference material for the PresentBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

PresentBlox Overview

PresentBlox provides a graphical user interface that can nest ChartBlox, GridBlox, PageBlox, ToolbarBlox, and DataLayoutBlox within a single presentation. Application assemblers use PresentBlox properties to tailor how these Blox appear.

PresentBlox makes extensive use of Blox qualifiers, as described in "Nested Blox" on page 6. For information on each nested Blox, refer to one of the following pages:

- "ChartBlox Overview" on page 65
- "DataLayoutBlox Overview" on page 211
- "GridBlox Overview" on page 215
- "PageBlox Overview" on page 269
- "ToolbarBlox Overview" on page 299

PresentBlox combines several Blox in one, providing users with simultaneous chart and grid views of the same data in the same window real estate.

Note: The user can click on a the Grid, Chart, Page Filter and Data Layout Panel buttons in the toolbar to show and hide these components in the PresentBlox. The user can also move the slider bar between the grid and chart to change the amount of space devoted to each view.

PresentBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a PresentBlox. For a copy and paste version of the tag with all the attributes, see "PresentBlox JSP Custom Tag" on page 465.

Syntax

```
<blox:present
    [attribute="value"] >
</blox:present>
```

where:

attribute	is one of the attributes listed in the attribute table.
value	is a valid value for the attribute.

Attribute
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
chartAvailable
chartFirst
dataLayoutAvailable
dividerLocation
enablePoppedOut
gridAvailable
height
helpTargetFrame
maximumUndoSteps
menubarVisible
noDataMessage
pageAvailable
poppedOut
poppedOutHeight
poppedOutTitle
poppedOutWidth
render
splitPane
splitPaneOrientation
toolbarVisible
visible
width

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:present>` tag with a closing slash (`/`) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is `width`, the end of the tag looks as follows:

```
width="650" />
```

Examples

```
<blox:present
   id="myPresent1"
   width="650"
   height="600"
   >
   <blox:data
      dataSourceName="TBC"
```

```
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year,
         Scenario) Qtr1 Qtr2 <CHILD Scenario Sales !"
   />
</blox:present>
```

PresentBlox Tag Attributes by Category

The following are PresentBlox tag attributes organized by categories of functionality. For lists of tag attributes common to several Blox, see "Common Blox Tag Attributes by Category" on page 29. The tag attributes supported by PresentBlox are organized in the cross reference as follows:

Data Area

The following tag attributes affect the data area of a PresentBlox:

- `dividerLocation`
- `splitPane`
- `splitPaneOrientation`

Chart Appearance

The following tag attributes affect the appearance of a ChartBlox inside the PresentBlox:

- `chartAvailable`
- `chartFirst`

Data Layout Appearance

The following tag attribute affects the appearance of the DataLayoutBlox on a PresentBlox:

- `dataLayoutAvailable`

Grid Appearance

The following tag attribute affects the appearance of a GridBlox on a PresentBlox:

- `gridAvailable`

Page Appearance

The following tag attribute affects the appearance of a PageBlox on a PresentBlox:

- `pageAvailable`

Menubar Appearance

The following tag attribute affects the appearance of a menu bar on a PresentBlox:

- `menubarVisible`

Toolbar Appearance

The following tag attribute affects the appearance of the toolbar on a PresentBlox:

- `toolbarVisible`

Popped Out Properties

The following are tag attributes regarding displaying PresentBlox in a separate, popped out browser window:

- `enablePoppedOut`
- `poppedOut`
- `poppedOutHeight`
- `poppedOutTitle`
- `poppedOutWidth`

PresentBlox Tag Attributes

This section describes the tag attributes supported by PresentBlox in alphabetical order. Common Blox tag attributes are listed but not described. For complete descriptions of common Blox tag attributes, see "Tag Attributes Common to Multiple Blox" on page 30. For reference information on PresentBlox methods, see the PresentBlox class in the com.alphablox.blox package in the Javadoc documentation.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 36.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see "applyPropertiesAfterBookmark" on page 30.

bloxEnabled

This is a common Blox property. For a complete description, see "bloxEnabled" on page 32.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 32.

chartAvailable

Specifies whether the chart is available to the user in PresentBlox.

Data Sources

All

Syntax

`chartAvailable="`*`available`*`"`

where:

Argument	Default	Possible Values
`available`	`true`	Specify `true` to make the chart available; `false` to make it unavailable.

Usage
The default is `true`. If set to `false`, the user cannot cause the chart to appear at all. To suppress the appearance of the chart until the user invokes it, use the dividerLocation property.

Examples
```
chartAvailable="false"
```

See Also
"chartFirst" on page 279, "dividerLocation" on page 280

chartFirst
Sets whether the chart appears before the grid when both appear in the PresentBlox display area.

Data Sources
All

Syntax
```
chartFirst="first"
```

where:

Argument	Default	Possible Values
`first`	`false`	Specify `true` to make the chart appear first; `false` to make it appear after the grid.

Usage
Depending on the value specified for the splitPaneOrientation property, "before" means "to the left of" or "above" the grid.

Examples
```
chartFirst="true"
```

See Also
"chartAvailable" on page 278, "chartFirst" on page 279, "dividerLocation" on page 280

dataLayoutAvailable
Specifies whether the data layout panel is available in PresentBlox.

Data Sources
All

Syntax
```
dataLayoutAvailable="available"
```

where:

Argument	Default	Possible Values
available	true	Specify true to make the data layout panel available; false to make it unavailable.

Usage

Note the following about the dataLayoutAvailable property:

- With this property set to false, the user cannot invoke the data layout panel. The Layout button will not appear on the toolbar.
- With this property set to true and the visible property set to false, the data layout panel appears only when the user clicks the toolbar layout button.

Examples

```
dataLayoutAvailable="false"
```

dividerLocation

Specifies where the available area should be divided into panes for displaying the chart and the grid.

Data Sources

All

Syntax

```
dividerLocation="location"
```

where:

Argument	Default	Possible Values
location	.5	A numeric value from 0 to 1 (of type double).

Usage

A value of 1 means that only the display on the left (or top, depending on the value of the splitPaneOrientation property) should appear. A value of 0 means that only the display on the right (or bottom) should appear. A value of .5 indicates that the area should be divided equally between the two displays.

Examples

```
dividerLocation=".7"
```

See Also

"chartAvailable" on page 278, "gridAvailable" on page 280, "splitPaneOrientation" on page 283.

enablePoppedOut

This is a property inherited from ContainerBlox. For a complete description, see "enablePoppedOut" on page 152.

gridAvailable

Specifies whether the grid is available in PresentBlox.

Data Sources
All

Syntax
```
gridAvailable="available"
```

where:

Argument	Default	Possible Values
`available`	`true`	Specify `true` to make the grid available; `false` to make it unavailable.

Examples
```
gridAvailable="false"
```

See Also
"dividerLocation" on page 280

height
This is a common Blox property. For a complete description, see "height" on page 35.

helpTargetFrame
This is a common Blox property. For a complete description, see "helpTargetFrame" on page 35.

maximumUndoSteps
This is a common Blox property. For a complete description, see "maximumUndoSteps" on page 36.

menubarVisible
This is a common Blox property. For a complete description, see "menubarVisible" on page 37.

noDataMessage
This is a common Blox property. For a complete description, see "noDataMessage" on page 37.

pageAvailable
Specifies whether the page panel is available in PresentBlox.

Data Sources
All

Syntax
```
pageAvailable="available"
```

where:

Argument	Default	Possible Values
`available`	`true`	When this value is set to `false`, page filters do not appear when a user drags items onto the Page axis on the Data Layout panel.

Examples

```
pageAvailable="false"
```

poppedOut

This is a property inherited from ContainerBlox. For a complete description, see "poppedOut" on page 153.

poppedOutHeight

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutHeight" on page 154.

poppedOutTitle

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutTitle" on page 154.

poppedOutWidth

This is a property inherited from ContainerBlox. For a complete description, see "poppedOutWidth" on page 155.

render

This is a common Blox property. For a complete description, see "render" on page 38.

splitPane

Specifies whether the data display area is split into panes once PresentBlox is instantiated.

Data Sources

All

Syntax

```
splitPane="split"
```

where:

Argument	Default	Possible Values
`split`	`true`	Specify `true` to split into panes; `false` to render a single data display area that chart and grid share.

Usage

The user toggles between the two data presentations using the Grid and Chart toolbar buttons.

See Also

"dividerLocation" on page 280, "splitPaneOrientation" on page 283

splitPaneOrientation

Specifies how to split the available area into panes for the chart and the grid.

Data Sources

All

Syntax

`splitPaneOrientation="orientation"`

where:

Argument	Default	Possible Values
`orientation`	`vertical`	Specify `vertical` to display Blox side-by-side (similar to portrait), or `horizontal` to display one Blox above the other (similar to landscape).

Usage

The value of the chartFirst property determines whether ChartBlox or GridBlox appears "first" (on the top or at the left).

Examples

```
splitPaneOrientation="horizontal"
```

See Also

"chartFirst" on page 279, "splitPane" on page 282

toolbarVisible

Specifies if the toolbar is visible.

Data Sources

All

Syntax

`toolbarVisible="visible"`

where:

Argument	Default	Description
`visible`	`true`	`true`: the toolbar is visible; `false`: the toolbar is not visible.

Usage

By default, the toolbar is visible in a PresentBlox. If a nested `<blox:toolbar>` tag is added, its setting overwrites the value of this attribute. For example, the following code will result in a visible toolbar.

```
<blox:present id="myPresent" toolbarVisible="false" ....>
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myDataBlox"/>
</blox:chart>
```

Tip: `toolbarVisible` is only a tag attribute, not a PresentBlox property.

visible

This is a common Blox property. For a complete description, see “visible” on page 40.

width

This is a common Blox property. For a complete description, see “width” on page 40.

Chapter 16. RepositoryBlox Tag Reference

This chapter contains reference material for the RepositoryBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

RepositoryBlox Overview

RepositoryBlox provides a means for developers to save and retrieve application properties and various objects stored in the DB2 Alphablox Repository. This capability is key to building a personalized application. Methods on RepositoryBlox fall into three categories:

- those for saving and maintaining multiple application states
- those providing access to application, user, and group properties
- those for saving and accessing objects stored in the DB2 Alphablox Repository

If multiple application states reside in the Repository, users can select the desired instance from the Applications page of the DB2 Alphablox Home Page.

Besides methods for saving and retrieving user, application, application state, and group properties, RepositoryBlox also provides methods for saving and retrieving Java objects of different types. These types are expressed as constants such as `TYPE_BINARY`, `TYPE_TEXT`, `TYPE_CONTAINER` (subfolders in the directory), `TYPE_HASHTABLE` (an array of objects), and `TYPE_XMLDOCUMENT`. This provides great flexibility and capability in the kind of data you can save and retrieve utilizing the DB2 Alphablox Repository.

Note: Group names are converted to all lowercase letters when stored in the repository to enhance performance.

RepositoryBlox has no graphical user interface. To invoke server-side RepositoryBlox methods, you can use the DHTML Client API, described in "Client-Side API Overview" on page 437.

RepositoryBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a RepositoryBlox. For a copy and paste version of the tag with all the attributes, see "RepositoryBlox JSP Custom Tag" on page 466.

Syntax

```
<blox:repository
    [attribute="value"] >
</blox:repository>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
bloxName
render

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting. You can substitute the closing </blox:repository> tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myRepositoryBlox" />
```

Examples

```
<blox:repository id="myRepository" />
```

RepositoryBlox Tag Attributes

This section describes the tag attributes supported by RepositoryBlox. All of the RepositoryBlox properties are common to multiple Blox and they are listed but not described in this section. For complete descriptions of common Blox properties, see "Tag Attributes Common to Multiple Blox" on page 30. For RepositoryBlox methods, see the RepositoryBlox class in the com.alphablox.blox package in the Javadoc documentation.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 36.

bloxName

This is a common Blox tag attribute. For a complete description, see "bloxName" on page 32.

render

This is a common Blox property. For a complete description, see "render" on page 38.

Chapter 17. ResultSetBlox Tag Reference

This chapter contains reference material for the ResultSetBlox. For general reference information about Blox, see Chapter 3, "General Blox Reference Information," on page 15. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

ResultSetBlox Overview

ResultSetBlox can be attached to a DataBlox to extend the normal functions associated with a JDBC data source. You can arbitrarily push a custom ResultSet into a DataBlox using ResultSetBlox. Or you can attach a method to the Blox to intercept queries in the associated DataBlox and to return arbitrary ResultSet objects to the DataBlox.

ResultSetBlox has `aresultSetHandler` property that lets you specify your result set handler class as follows:

```
<blox:data id="rsData"
   dataSourceName="canned"
   connectOnStartup="false"
/>
...
<blox:resultSet id="rset1"
   dataBlox="<%=rsData%>"
   resultSetHandler="<%=new TupleResultSet()%>"
/>
```

where `rsData` is a previously defined DataBlox, and TupleResultSet is your result set handler. This handler should implement the IResultSetHandler interface, which provides an `executeQuery()` method that returns a java.sql.ResultSet object, and a `fetchComplete()` method that terminates the connection when the data from the result set is fetched:

```
<%@ page import="com.alphablox.blox.*,
                 java.sql.*" %>
...

<%!
 public class TupleResultSet implements IResultSetHandler {
    ...
    // Store your custom query into the String myQuery

    Connection conn = null;
    public ResultSet executeQuery(String myQuery) throws Exception {
       //code here to get connected
    }

    public void fetchComplete() throws Exception {
       //close the connection
       conn.close();
       conn = null;
    }
%>
```

Note the following:

- The DataBlox in the above example does not initially connect to any data source (`connectionOnStartup = "false"`). This is a useful technique in cases where we do not want to get the initial result set, and we want users to make some selections in order for us to construct the query string dynamically and to issue the query using JDBC connection.
- You need to add the import statement for java.sql.*.
- Depending on the type of data you are getting from the result set, there is a minimum set of APIs you need to implement on java.sql.ResultSet. These APIs are listed in the next section.

For a complete live example, see the Blox Sampler's Retrieving Data section.

Minimum APIs to Implement on ResultSet

The following is a list of the minimum set of APIs you need to implement on java.sql.ResultSet, depending on the type of data you are getting from the result set.

- `next()` : void
- Getter methods to implement depending on the data types your result set returns:
  - `getInt(int)`: Integer
  - `getBoolean(int)`: Boolean
  - `getBigDecimal(int)`: BigDecimal
  - `getFloat(int)`: Float
  - `getDouble(int)`: Double
  - `getString(int)`: String
  - `getDate(int)`: Date
  - `getObject(int)`: Object
  - `getMetaData()`: java.sql.ResultSetMetaData

If the returned result set is of type java.sql.ResultSetMetaData, you should implement the following methods:

- `getColumnCount()` : int
- `getColumnType(int)` : int
- `getScale(int)` : int
- `getPrecision(int)` : int
- `getColumnName(int)` : String
- `getColumnLabel(int)`: String
- `getColumnTypeName(int)` : String
- `getColumnType(int)` : int

ResultSetBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a ResultSetBlox. For a copy and paste version of the tag with all the attributes, see "ResultSetBlox JSP Custom Tag" on page 466.

Syntax

```
<blox:resultSet
    [attribute="value"] >
</blox:resultSet>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
bloxName
dataBlox
resultSetHandler

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting. You can substitute the closing `</blox:resultSet>` tag using the shorthand notation, closing the tag at the end of the attribute list that looks as follows:

```
id="myResultSet" />
```

Examples

```
<blox:resultSet id="myResultSet" />
```

ResultSetBlox Tag Attributes

This section describes the tag attributes supported by ResultSetBlox. Tag attributes common to multiple Blox are listed but not described in this section. For complete descriptions of common Blox properties, see "Tag Attributes Common to Multiple Blox" on page 30. For ResultSetBlox methods, see the ResultSetBlox class in the com.alphablox.blox package in the Javadoc documentation.

id

This is a common Blox tag attribute. For a complete description, see "id" on page 36.

bloxName

This is a common Blox tag attribute. For a complete description, see "bloxName" on page 32.

dataBlox

The DataBlox associated with this ResultSetBlox.

Data Sources

Relational

Syntax

```
dataBlox="dataBlox"
```

where:

Argument	Default	Description
`dataBlox`		The DataBlox this ResultSetBlox is attached to.

resultSetHandler

The handler that implements the `executeQuery()` method to take a query and return a result set.

Data Sources

Relational

Syntax

`resultSetHandler="`*`handler`*`"`

where:

Argument	Default	Description
`handler`		The handler to implement in order to execute a query.

Usage

This handler needs to implement the methods in IResultSetHandler. See the IResultSetHandler interface in the com.alphablox.blox package in the Javadoc documentation.

Chapter 18. StoredProceduresBlox Tag Reference

This chapter contains reference material for StoredProceduresBlox for using stored procedures.

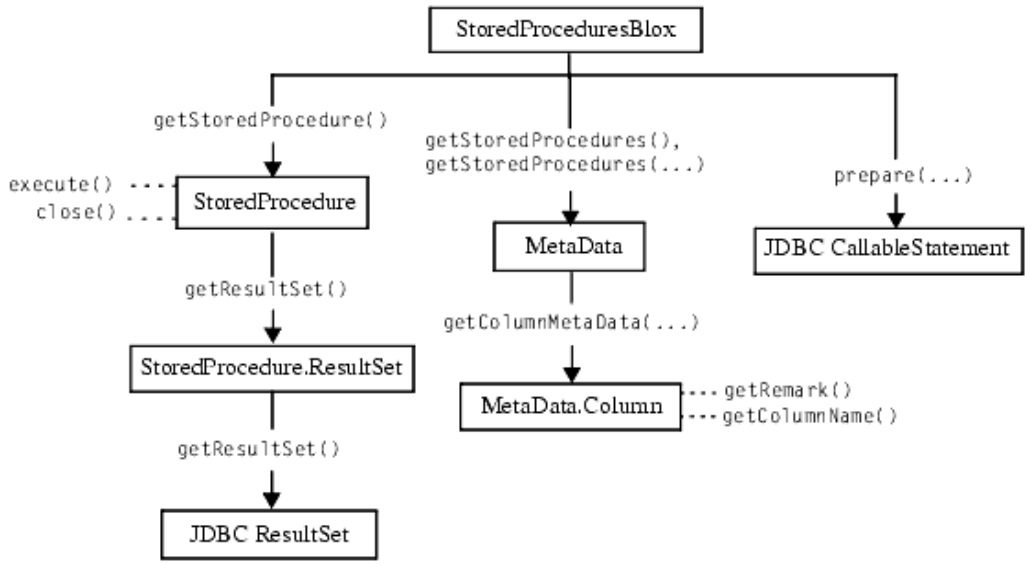
StoredProceduresBlox Overview

StoredProceduresBlox is the starting point for using relational database stored procedures. It allows you to create a connection to a database and prepare a stored procedure statement. Once the correct DB2 Alphablox data source and any other connection parameters are set, you can:

- use the `prepare(...)` method to return a JDBC CallableStatement object, which can be used to set up any stored procedure parameters necessary to execute the stored procedure
- use the `getStoredProcedure()` method to access the current StoredProcedure object; you can then execute the stored procedure, get to the ResultSet of the executed stored procedure, or access the JDBC ResultSet
- use the `getStoredProcedures()` or `getStoredProcedures(...)` methods to return one or more MetaData objects that give you access to the individual parameters

The StoredProcedure object and the MetaData object are separate classes in the `com.alphablox.blox.data.rdb.storedprocedure` package. By having separate objects for StoredProcedure and MetaData from StoredProceduresBlox, you can prepare a stored procedure once and then execute it multiple times. Even though stored procedure parameters can be altered between executions, you can enhance the performance by not preparing the stored procedures at every execution.

The following diagram shows the object hierarchy of stored procedure related objects.



Because the StoredProcedure and MetaData objects are in a separate package, you must use the following JSP import statement at the beginning of any JSP file to use any of the APIs in these objects:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
```

Note: JDBC Stored procedures are supported for IBM DB2 UDB, Sybase, Oracle, and Microsoft SQL Server databases.

Note the following when using the StoredProcedure object to execute a prepared stored procedure:

- If a DataBlox is used to display information from a stored procedure, the DataBlox must be separately connected to the same data source as StoredProceduresBlox.
- If a DataBlox is used to display information from a stored procedure and the stored procedure also has output parameters, the result set must first be used before getting the output parameters. This is a JDBC restriction.
- If the stored procedure has input and output parameters, you should use `StoredProceduresBlox.prepare(...)` to get the JDBC CallableStatement object. This object allows you to get and set input and output parameters on the stored procedure.
- Once the stored procedure has been executed and any output parameters or result sets are used, you need to call the `StoredProceduresBlox.disconnect()` to disconnect and free up any resources. If you want to keep the connection to the data base open, call `StoredProceduresBlox.close()` to free up any resources used.
- If a DataException is thrown, extra information might be available as a SQLException by looking at `DataException.getNestedException()`.

Once the stored procedure is executed, it returns a StoredProcedure.ResultSet object, which gives you access to the JDBC ResultSet object. If you need to use the JDBC ResultSet object directly, use the `ResultSet.getResultSet()` method to get to this object.

It is recommended that you also import the java.sql package when working with stored procedures, so your JSP files should import two packages:

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
```

StoredProceduresBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a StoredProceduresBlox.

Syntax

```
<blox:storedProcedures
    [attribute="value"] >
</blox:storedProcedures>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
bloxName

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:storedProcedures>` tag using the shorthand notation: closing the tag at the end of the attribute list that looks as follows:

```
id="myBlox" />
```

Examples

```
<blox:storedProcedures
     id="namedStoredProceduresBlox" />
```

StoredProceduresBlox Examples

This section includes six examples that demonstrates the use of StoredProceduresBlox and its associated objects. For more examples, see the Javadoc documentation.

Example 1: Connecting to the data source without a DataBlox

This example demonstrates how to connect to the data source without a DataBlox as you may only want to get the parameters or run an INSERT SQL stored procedure that does not require a DataBlox.

```
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox" %>

<blox:storedProcedures id="mySP"/>
<%
    mySP.setDataSourceName("sales");
    mySP.connect();
%>
```

Example 2: Using the StoredProceduresBlox to connect the data source for use with DataBlox

This example demonstrates how the DataBlox used to display information from a stored procedure needs to be separately connected to the same data source as StoredProceduresBlox.

```
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>

<blox:data id="myDataBlox" visible="false"/>

<%
    myDataBlox.setDataSourceName("sales-sql");
    myDataBlox.connect();
    mySP.setDataSourceName("sales-sql");
    mySP.connect();
%>
```

Example 3: Getting a list of stored procedures whose name matches a specified pattern

This example demonstrates how to use the `getStoredProcedures(...)` method to get a list of stored procedures whose name starts with ″procedure″. This method returns an array of MetaData objects. The MetaData object contains information on the parameters for each stored procedure.

```
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>
<%
    mySP.setDataSourceName("sales-sql");
    mySP.connect();
    MetaData procedures[] =
        mySP.getStoredProcedures("procedure%");
%>
<%
   if (procedures.length == 0) {
%> <strong>No procedures found.</strong> <%
} %>
```

Through the MetaData object, you can then access the individual parameter for a specified stored procedure.

Example 4: Getting a list of all parameters for each stored procedure

This example demonstrates how to use the MetaData object to get to each stored procedure and the parameters for each stored procedure. This example assumes you already have a MetaData object returns as shown in the previous example:

```
MetaData procedures[] =
        mySP.getStoredProcedures("procedure%");
```

We will now list each stored procedure and its catalog, schema, name, and remark information in a table:

```
<table border="1" >
<tr><th colspan="4">Stored Procedure Information</th></tr>
<tr><th>Catalog</th><th>Schema</th><th>Name</th><th>Remarks</th></tr>
<%
   for (int i = 0; i < procedures.length; i++) {
      String catalog = procedures[i].getCatalog();
      String schema = procedures[i].getSchema();
      String name = procedures[i].getName();
      String rem = procedures[i].getRemark();
      String type = null;
   %>
   <tr><td><%= catalog %></td>
       <td><%= schema %></td>
       <td><%= name %></td>
       <td><%= rem %></td></tr>
   <%
   }
%>
</table>
```

We can also get the detail of each parameter for each stored procedure:

```
//for each of the stored procedure, we will get the MetaData.Column //
object which contains the detail of the parameters
<%
for (int spCount = 0; spCount < procedures.length; spCount++) {
   String currProcedure = procedures[spCount].getName();
   MetaData.Column cMeta[] =    procedures[spCount].getColumnMetaData();%>

   //for the current stored procedure, we will get the list the
   //detail for each parameter in a table

   <table border="1">
   <tr><th colspan="7">Stored Procedure Params for
   <%=currProcedure %></th></tr>
   <tr><th>Catalog</th><th>Schema</th><th>Name</th><th>Column Name</
th><th>Type</th><th>Type Name</th><th>Remark</th></tr>

   //Iterate through the parameters in the current stored procedure
   <% for (int i = 0; i < cMeta.length; i++) {
       String catalog = cMeta[i].getCatalog();
       String schema = cMeta[i].getSchema();
       String name = cMeta[i].getName();
       String colName = cMeta[i].getColumnName();
       short type = cMeta[i].getType();
       String typeName = cMeta[i].getTypeName();
         String remark = cMeta[i].getRemark();
   %><tr><td><%= catalog %></td>
         <td><%= schema %></td>
         <td><%= name %></td>
```

```
        <td><%= colName %></td>
        <td><%= type %></td>
        <td><%= typeName %></td>
        <td><%= remark %></td></tr><%
   } %>
   </table>
<% }
%>
```

Example 5: Executing a stored procedure that has one input parameter and two output parameters

This example demonstrates how to the `prepare()` method to return a JDBC CallableStatement object that you can use to execute a stored procedure with input and output parameters.

```
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="com.alphablox.blox.data.rdb.*" %>
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>

<%
    mySP.setDataSourceName("storeSales");
    mySP.connect();

    // param 1 is an integer output, param 2 is a string input,
    // param 3 is a string output
    CallableStatement cstmt = mySP.prepare("{call a_procedure(?, ?, ?)}");
    cstmt.setString(2, "users/admin%");
    cstmt.registerOutParameter(1, Types.INTEGER);
    cstmt.registerOutParameter(3, Types.VARCHAR);
    mySP.execute();
    int out1 = cstmt.getInt(1);
    String out3 = cstmt.getString(3);
%>
...

<!-- Closes all resources associated with executing the stored procedure -->
<%
    mySP.close();
%>
...
<!--Disconnects from the data source -->
<%
    mySP.disconnect();
%>
```

Example 6: Setting a stored procedure result set to a DataBlox

This example demonstrates how to get a stored procedure result set to a DataBlox.

```
<%@ page import="com.alphablox.blox.data.rdb.*" %>
<%@ page import="com.alphablox.blox.data.rdb.storedprocedure.*" %>
<%@ page import="com.alphablox.blox.StoredProceduresBlox" %>
<%@ page import="java.sql.*" %>
<%@ taglib uri="bloxtld" prefix="blox"%>

<blox:storedProcedures id="mySP"/>

<blox:data id="myDataBlox" visible="false" />
<%
    myDataBlox.setDataSourceName("sales-sql");
```

```
    myDataBlox.connect();
    mySP.setDataSourceName("sales-sql");
    mySP.connect();
    mySP.prepare("{call a_procedure}");
    mySP.execute();
    mySP.loadResultSet(myDataBlox, 1);
%>
```

StoredProceduresBlox Tag Attributes

This section describes the tag attributes supported by StoredProceduresBlox. For StoredProceduresBlox methods, see the StoredProceduresBlox class in the com.alphablox.blox package in the Javadoc documentation.

id

This is a common Blox tag attribute. For a complete description, see “id” on page 36.

bloxName

This is a common Blox tag attribute. For a complete description, see “bloxName” on page 32.

Chapter 19. ToolbarBlox Tag Reference

This chapter contains reference material for ToolbarBlox. For general reference information about Blox, see Chapter 3, “General Blox Reference Information,” on page 15. For information on how to use this reference, see Chapter 1, “Using This Reference,” on page 1.

ToolbarBlox Overview

ToolbarBlox presents a customized Blox toolbar. It is added in two ways:

- Using the nested `<blox:toolbar>` tag inside a PresentBlox, ChartBlox, or GridBlox.
- Setting the `toolbarVisible` tag attribute to `true` for `<blox:present>`, `<blox:chart>`, or `<blox:grid>` tag.

In the DHTML client, you cannot have a standalone ToolbarBlox. By default, the toolbar is available and visible in a PresentBlox, a standalone GridBlox, and a standalone ChartBlox.

Graphical User Interface

ToolbarBlox appears in the DHTML client with two toolbars: Standard toolbar and Navigation toolbar. By default, these two toolbars contain the following buttons:

- Pop out
- Copy
- Redo
- Undo
- Load Bookmark
- Export to PDF
- Export to Excel
- Help
- Data Navigation
- Sort
- Member Filter
- Grid
- Chart
- Page Filter
- Data Layout Panel

The buttons and the toolbars are fully customizable. The Blox UI Tag Library includes tags that allow you to add, edit, or remove a toolbar or a toolbar button. See Chapter 23, “Blox UI Tag Reference,” on page 367 for details.

For instructions on using the ToolbarBlox user interface, see the online user help. You can access the user help by clicking the help button on the toolbar of the Blox user interface.

ToolbarBlox JSP Custom Tag Syntax

The DB2 Alphablox Tag Libraries provides custom tags to use in a JSP page for creating each Blox. This section describes how to create the custom tag to create a toolbar within a PresentBlox, GridBlox, or ChartBlox. For a copy and paste version of the tag with all the attributes, see "Miscellaneous Tags in blox.tld" on page 467.

Parameters

```
<blox:toolbar
    [attribute="value"] >
</blox:toolbar>
```

where:

attribute is one of the attributes listed in the attribute table.

value is a valid value for the attribute.

Attribute
id
applyPropertiesAfterBookmark
bloxEnabled
bloxName
bookmarkFilter
helpTargetFrame
removeAction
removeButton
rolloverEnabled
textVisible
toolTipsVisible
visible

Usage

Each custom tag can have one or more attributes, separated by one or more space or new line characters. Any extra space or new line characters are ignored. For readability, attributes can each go on a separate line with the same indenting.

You can substitute the closing `</blox:toolbar>` tag with a closing slash (`/`) after the last attribute in the tag but before the closing greater than character. For example, if the last attribute is `width`, the end of the tag looks as follows:

```
width="650" />
```

Examples

```
<blox:toolbar
   id="myToolbar1"
   toolTipsVisible="false">
</blox:toolbar>
```

ToolbarBlox Tag Attributes by Category

The following are unique ToolbarBlox tag attributes. For tag attributes common to several Blox, see "Common Blox Tag Attributes by Category" on page 29. The tag attributes supported by ToolbarBlox are organized in the cross reference as follows:

- "Appearance" on page 301
- "Contents" on page 301

Appearance

The following tag attributes affect the appearance of a ToolbarBlox:

- `rolloverEnabled`
- `textVisible`
- `toolTipsVisible`

Contents

The following tag attribute affects the contents of a ToolbarBlox:

- `removeButton`

ToolbarBlox Tag Attributes

This section describes the tag attributes supported by ToolbarBlox. Common Blox properties available from ToolbarBlox are listed but not described. For complete descriptions of common Blox properties, see "Tag Attributes Common to Multiple Blox" on page 30.

id

This is a common Blox property. For a complete description, see "id" on page 36.

applyPropertiesAfterBookmark

This is a common Blox property. For a complete description, see "applyPropertiesAfterBookmark" on page 30.

bloxEnabled

This is a common Blox property. For a complete description, see "bloxEnabled" on page 32.

bloxName

This is a common Blox property. For a complete description, see "bloxName" on page 32.

bookmarkFilter

This is a common Blox property. For a complete description, see "bookmarkFilter" on page 31.

helpTargetFrame

This is a common Blox property. For a complete description, see "helpTargetFrame" on page 35.

removeAction

This is a common Blox property. For a complete description, see “removeAction” on page 38.

removeButton

Identifies the buttons to remove from the ToolbarBlox (before it appears to the user).

Data Sources

All

Syntax

```
removeButton = "removeButton"
```

where:

Argument	Default	Description
removeButton	"Save,Load"	Comma-delimited string of button names.

Usage

The value is a quoted, comma-delimited list of button names, such as “Save, Load”, which removes those buttons from the toolbar, thus removing user access to the Save/Load application state function. The possible values for the button names are Chart, Layout, Grid, Swap, Bookmark, Help, Load, Save.

Tip: The list you supply in `setRemoveButton()` method will overwrite the default. If you do not want the Save and Load buttons, make sure you include them in your list of buttons to remove. To remove buttons not listed here, use the Blox UI tags. See “Custom Toolbar Tags” on page 400.

Tip: The Save and Load buttons allow a user to save the state of the application as private or public in much the same way as the bookmark functionality. The difference is, when you have multiple presentation Blox that are not nested, the Save and Load buttons will save the state of all Blox on the page automatically and you do not need to specify which Blox you want to save the state, as is the case with the bookmark functionality. The saved application states are managed separately from bookmarks, so you may want to avoid confusion by offering only the bookmark or the save/load application state functionality.

Examples

```
removeButton="Chart,Save,Load"
```

rolloverEnabled

Specifies whether the color of toolbar buttons should change from grayscale to color when the mouse moves over the button.

Data Sources

All

Syntax

```
rolloverEnabled = "boolean"
```

where:

Argument	Default	Description
enable	false	A boolean argument. A value of true indicates the toolbar button image changes on a mouse over to show a rollover effect; a value of false indicates the toolbar button image does not change.

Usage
Toolbar buttons appear with a rollover effect with mouse over if this property value is set to true. If you add a toolbar button using the <bloxui:toolbarButton> tag, you need to supply an image with the "_active" suffix for the mouse-over effect when this property is set to true. See "Custom Toolbar Tags" on page 400 for more information.

Examples
```
rolloverEnabled="false"
```

textVisible
Specifies whether a text label should appear beneath the icon on a toolbar button.

Data Sources
All

Syntax
```
textVisible = "boolean"
```

where:

Argument	Default	Description
visible	false	A boolean argument. A value of true indicates the text labels are visible, a value of false indicates they are not.

Usage
A text label appears beneath the icon on a toolbar button when the value is set to true.

Examples
```
textVisible="false"
```

toolTipsVisible
Specifies whether descriptive text should appear when the user holds the mouse over a toolbar button.

Data Sources
All

Syntax
```
toolTipsVisible = "boolean"
```

where:

Argument	Default	Description
`visible`	`true`	A boolean value. A value of `true` indicates the tooltips display when you mouse over the toolbar, a value of `false` indicates the tooltips do not display.

Usage

Descriptive text appears when the user holds the mouse over a toolbar button when the value is set to `true`.

Examples

"rolloverEnabled" on page 302, "toolTipsVisible" on page 303

visible

This is a common Blox property. For a complete description, see "visible" on page 40.

Chapter 20. Blox Form Tag Reference

There are many variations of FormBlox that allow you to add HTML form-like user interface in your JSP and link the form elements to server-side components or other form components on the page without page refreshes. The tags to add these FormBlox are available in the Blox Form Tag Library (`bloxform.tld`). This chapter contains reference material for tags in this library. For detailed API listing, see the com.alphablox.blox.form package in the Javadoc documentation.

FormBlox Overview

FormBlox and business logic Blox (discussed in Chapter 21, "Business Logic Blox and TimeSchema DTD Reference," on page 341) are designed to solve two commonly encountered problems during analytical application development: the need for data-aware business logic and the need to maintain state. A series of specialized FormBlox let you generate time periods, data source, cube, dimension, and member selection lists simply by using the Blox Form Tag Library (`bloxform.tld`).

- These Blox let you create user interfaces similar to those standard HTML form elements such as radio buttons, check boxes, and edit fields.
- Unlike generic HTML form elements, FormBlox automatically maintain the state after page reloads during a session.
- FormBlox can automatically populate a selection list based on the data source, cube, dimension, or time schema you specify.

As a result, there is no need to write code to perform sophisticated time series calculation, find out the metadata in order to populate a user selection list, or to manage the state of the form elements.

FormBlox Variations

There are different variations of FormBlox, each designed to add a specific user interface that can be linked to other FormBlox or server-side components. The following tables lists all FormBlox and describes their purposes:

FormBlox	Description
FormBlox for Generic HTML Form Elements	
CheckBoxFormBlox	A FormBlox implementation of the HTML <input type="checkbox"...> tag. However, unlike an HTML form, which, when posted, does not send any value if a box is not checked, CheckBoxFormBlox always returns a value on a form post.
EditFormBlox	A FormBlox implementation of an HTML edit field (either <input type="text" ...> or <textarea ...>)
RadioButtonFormBlox	A FormBlox implementation of the HTML radio button set (<input type="radio" ... >).
SelectFormBlox	A FormBlox implementation of an HTML <select> element; supports both single and multiple select.
FormBlox for MetaData Selection List	
DataSourceSelectFormBlox	A FormBlox implementation of an HTML <select> list that displays available data sources; can be limited to multidimensional or relational data sources.
CubeSelectFormBlox	A FormBlox implementation of an HTML <select> list that displays cubes available in a given multidimensional data source.
DimensionSelectFormBlox	A FormBlox implementation of an HTML <select> list that displays dimensions available in a given multidimensional cube.
MemberSelectFormBlox	A specialized implementation of the HTML <select> element that displays members from a given multidimensional data source dimension. Given a DataBlox, a cube (if necessary), and a dimension, it displays a selection list containing members within the dimension.
FormBlox for TimeSchema-Related Selection List	
TimePeriodSelectFormBlox	A FormBlox implementation of the HTML <select> element that displays TimeSeries available in a time schema. A TimeSeries is a duration of time such as last two months, last quarter, last two quarters, month to current, quarter to current, current month, and current week.
TimeUnitSelectFormBlox	A specialized implementation of the HTML <select> element that displays period types from a given time schema. A time unit (PeriodType) is the unit used by a time schema, such as years, halves, quarters, months, weeks, and days.
FormBlox for Navigation Tree	
TreeFormBlox	A FormBlox encapsulation of the Blox UI tree control. A DHTML tree control is rendered to the page.

All FormBlox-related classes are under the com.alphablox.blox.form package. Their tags are available from the `bloxform.tld` tag library.

Common FormBlox Properties and Attributes

The FormBlox class is the base class for all FormBlox. As such, all FormBlox share some common properties, methods, tags, and behavior:

- They use the same event model FormEventListener. This is how all FormBlox events are handled.
- They use a form POST to post values (except TreeFormBlox).
- They all have the following tag attributes:

Common FormBlox Attribute	Description
`id`	The `id` of the object that is rendered on the page. It is also the `bloxName` unless the `bloxName` attribute is specified.
`bloxName`	The name of the object on the server (the peer).
`formElementName`	The name of the rendered page element and the parameter name used for a form POST.
`themeClass`	The name of a theme class.
`visible`	`true` if the object is rendered in place. The default is `true`.

FormBlox Events

The FormEventListener interface in com.alphablox.blox.form is the event handler for all FormBlox. The `addFormEventListener()` and `removeFormEventListener()` methods allow you to add/remove a FormEventListener to enable/disable event handling. When event handling, the `FormEventListener.valueChanged()` method is called whenever a FormBlox is changed (for example, a check box is checked/unchecked, a radio button is clicked, or a selection is made in a selection list).

The `getChangedProperty` and `setChangedProperty` tags support basic event handling and, in many cases, can make writing an event handler unnecessary. The cases where a `getChangedProperty` and `setChangedProperty` tags will suffice are those where a property on one object will always change a corresponding property on another Java bean.

The setChangedProperty Tag

FormBlox can set a property on any Java bean. The `<bloxform:setChangedProperty>` tag lets you specify which property should be changed to the new value selected when a FormBlox is changed. Consider a check box that allows a user to choose whether to enable alternate row colors on a grid. The checkbox is added using the CheckBoxFormBlox:

```
<bloxform:checkBox id="bandingCheckBox"
   checked="false"
   checkedValue="true"
   uncheckedValue="false">
   <bloxform:setChangedProperty
      targetRef="myGridBlox"
      targetProperty="bandingEnabled" />
</bloxform:checkBox> Enable alternate row banding
```

This check box is unchecked as it is rendered on the page. When users check the box, the checked value (`checkedValue`) will be set on the `bandingEnabled` property (`targetProperty`) of `myGridBlox` (`targetRef`).

The getChangedProperty Tag

The <bloxform:getChangedProperty> tag is nested in the tag for a FormBlox and establishes that a property on the FormBlox will change whenever a corresponding property on another FormBlox changes. For example, it lets you link several FormBlox so the selection from one FormBlox sets the selection available in another FormBlox. A common scenario is the so-called "cascading menus." In cascading menus, the selection of an option from the first menu dictates the options available in the next menu.

For example, in a report we may have a set of menus for users to select a city to see sales data. The cascading menus start from a zone menu. The selection of a zone dictates the areas available in the second menu. The selection of an area dictates the cities available in the subsequent menu. The following example creates the zone menu by getting generation 2 members of the All Locations dimension. The area menu is created by getting the selected member from the zone menu. The following is the code snippet:

```
<%--The zone menu: displaying all generation 2 members of
    the All Locations dimension. Note that for MSAS data sources
    the member name should be enclosed in square brackets (unique
    names). --%>
<bloxform:memberSelect id="zone"
   dataBloxRef="myData"
   dimensionName="All Locations"
   filterOperator="=="
   filterGeneration="2">
</bloxform:memberSelect>

<%--The area menu: displaying all generation 3 members of
    the selected member from the zone menu. --%>
<bloxform:memberSelect id="area"
   dataBloxRef="myData"
   dimensionName="All Locations"
   filterOperator="=="
   filterGeneration="3">
   <bloxform:getChangedProperty
      formBloxRef="zone"
      formProperty="selectedMembers"
      property="rootMembers" />
</bloxform:memberSelect>
```

The FormPropertyLink Object

The FormPropertyLink class in the com.alphablox.blox.form package is the object behind the `getChangedProperty` and `setChangedProperty` tags. It is used to link FormBlox together and transfer basic properties back and forth between them.

Whenever a property changes on a FormBlox, FormPropertyLink will set the new value on the target bean. Since it is using the normal Java bean introspection, the target can be any Java beans rather than just FormBlox. If necessary, one additional method on the bean can be called following the property change. This allows the link to handle cases such as a change to the `query` property on the DataBlox. In this case, in order to complete the change, it is necessary to call the DataBlox's `updateResultSet()` method following a change to the query property.

FormPropertyLink will automatically perform conversions of different data types when you use the tags to link two different properties:

- If the caller's argument and the callee's expected parameter are of the same type, then the argument is passed to the callee as is.

- If the caller's argument is an array and the callee's expected parameter is not, then the first element in the passed array will be passed to the callee.
- If the caller's argument is not an array and the callee is expecting an array, then the argument is converted into an array of length 1 to be passed to the callee.
- If the caller's argument is a String and the callee is expecting a boolean, then the string will be converted to a boolean. For example, the string `"true"` will be converted to a boolean `true` and passed to the callee.
- If the caller's argument is a non-primitive Java object and the callee is expecting a String, then the argument will be converted to a String using `toString()` before being passed to the callee.

Styling FormBlox

All FormBlox have a `themeClass` property and a `themeClass` tag attribute for you to specify a theme class for the component. The following TreeFormBlox uses a style class called `myMenuTree` to set the style for the menu item texts:

```
<!--some code ommitted here...>
<head>
   <blox:header/>
   <style>
   .myMenuTree { background-color: #FFFF80; }
   </style>
</head>
<body>
<bloxform:tree id="myMenu" rootVisible="false" themeClass="myMenuTree">
   <bloxform:folder> <%--root folder--%>
      <bloxform:folder label="Sales Analysis">
         <bloxform:item label="Sales Trend by Region"
            href="salesByRegion.jsp"
            target="mainFrame" />
         <bloxform:item label="Sales by Store"
            href="salesByStore.jsp"
            target="mainFrame" />
         <bloxform:item label="Units Sold by Product"
            href="unitsSoldByProduct.jsp"
            target="mainFrame" />
      </bloxform:folder>
   </bloxform:folder>
<!--more code omitted here-->...
```

You can also use the DB2 Alphablox theme classes defined in the `<themeName>_dhtml.css` file in `<alphablox_dir>/repository/theme/<themeName>`. This allows for a consistent look and feel through out your application. The following example shows a SelectFormBlox that uses a defined theme class called `csSlctBg`.

```
<!--some code omitted here-->...
<b>Select Chart Type:</b><br>
<bloxform:select id="ChartSelection" size="4"
   themeClass="csSlctBg">
   <bloxform:option label="Bar" value="Bar" selected="true"/>
   <bloxform:option label="Pie" value="Pie" />
   <bloxform:option label="Line" value="Line" />
   <bloxform:option label="3D Bar" value="3D Bar" />
   <bloxform:setChangedProperty
      targetRef="myChart"
      targetProperty="chartType" />
</bloxform:select>
<!--more code omitted here-->......
```

For details on how CSS themes are supported and used, see the Presenting Data chapter of the *Developer's Guide*. It also includes a listing of style classes supported in DB2 Alphablox themes.

FormBlox that Create a Selection List

Many FormBlox create a selection list. All the different variations of SelectFormBlox behave similarly as the first option in the list is the default selected option if this is a single selection list and no selected option is set explicitly. With the exception of DataSourceSelectFormBlox and TimePeriodSelectFormBlox, these Blox have a `multipleSelect` tag attribute and a `size` tag attribute. When the selection list has a `size` greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur. Since most of these Blox are tied to a DataBlox and their instantiation involves a query to the data source, an initial selection should be set.

Note: With FormBlox that are tied to a DataBlox, every time a selection is made on the selection list, the `FormEventListener.valueChanged()` method is called and a query is issued. When working with large result set or complex queries, there could be delay or performance issues.

Blox Form Tag Library Reference by Category

To use the following FormBlox tags, include the following taglib directive in the beginning of your JSP files:

```
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
```

For FormBlox methods, see the com.alphablox.blox.form package in the Javadoc documentation.

The Blox Form Tag Library includes the following form tags:

FormBlox for Generic HTML Form Elements

FormBlox for Data-Related Selection List

FormBlox for TimeSchema-Related Selection List

TreeFormBlox

Nested Tag for Connecting FormBlox and Specifying Actions

The following sections describe the properties, tags, and attributes for each of the FormBlox, with examples that demonstrate the usage and syntax.

CheckBoxFormBlox Reference

For each check box you add, you can specify whether this check box should be checked when it is rendered, and what value should be passed when the box is checked or unchecked. For better page layout, you may want to add each CheckBoxFormBlox inside a table cell in order to put text next to it. Note that as soon as users click the check box, the `valueChanged()` method on the FormEventListener is called and the new value is set immediately.

CheckBoxFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for CheckBoxFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
`checked`	`boolean`	`true` if the check box should be checked and the `checkedValue` should be set when the check box is rendered. The default is `false`.
`checkedValue`	`String`	The value to return when the check box is checked. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The posted value of the component.
`formValues`	`String[]`	The posted values of the component.
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`themeClass`	`String`	A String containing the theme class name(s) to set on this element. Separate class names with a space.
`title`	`String`	A String containing the text to display after the check box.
`uncheckedValue`	`String`	The value to return when the check box is unchecked. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.

The <bloxform:checkBox> Tag

The following table lists all attributes for the `<bloxform:checkBox>` tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
checked	false	true if the check box should be checked and the checkedValue should be set when the check box is rendered. If this attribute is not specified, the check box is unchecked and the uncheckedValue will be set when the check box is rendered.
checkedValue	true	The value to return when the check box is checked. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
title		A String containing the text to display after the check box.
uncheckedValue	false	The value to return when the check box is unchecked. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
visible	true	true if the object is to be rendered in place.

A CheckBoxFormBlox Example

This example demonstrates how to allow users to turn on/off alternate row banding in a GridBlox.

- A GridBlox is added, but is not rendered (visible="false").
- A CheckBoxFormBlox is added with the text to display after it defined in the title attribute.
- The checked attribute is set to true. Therefore, the check box, when rendered on the page, is checked and the checkedValue is set on myGridBlox's bandingEnabled property. Note that the checkedValue attribute is not specified, so the default value "true" is used.
- The nested <bloxform:setChangedProperty> tag is used to specify the target object and the property of the object to change.
- The GridBlox is rendered using the <blox:display> tag, and the GridBlox is displayed with the alternate row banding enabled.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>

<blox:grid id="myGridBlox"
   visible="false"
   width="600"
   height="350">
   <blox:data
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW (\"All Products\") <CHILD \"All Products\"
        <COL (\"All Time Periods\") <CHILD \"All Time Periods\"
         <PAGE(Measures) Sales !" />
</blox:grid>

<html>
<head>
   <blox:header />
</head>

<body>
<bloxform:checkBox id="bandingCheckBox"
   title="Enable Alternate Row Banding"
   checked="true">
```

```
   <bloxform:setChangedProperty
      targetRef="myGridBlox"
      targetProperty="bandingEnabled" />
</bloxform:checkBox>

<blox:display bloxRef="myGridBlox" />

</body>
</html>
```

CubeSelectFormBlox Reference

This FormBlox adds a selection list of cubes available in a given multidimensional data source.

CubeSelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for CubeSelectFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
dataBlox	DataBlox	The DataBlox from which to get the list of cubes.
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedCube	Cube	The selected Cube object.
selectedCubeNames	String[]	The names of the selected cubes.
selectedCubes	Cube[]	The selected Cube objects.
size	int	The number of lines that are visible in the list. The default is 1. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

Note: Most FormBlox that create a selection list (CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox) have the same behavior: when the selection list has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this selectedCube/Dimension/Member/Series attribute

is explicitly specified. When the selection list has a `size` greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

The <bloxform:cubeSelect> Tag

The following table lists all attributes for the `<bloxform:cubeSelect>` tag:

Attribute	Default	Description
`id`		The `id` of the object that is rendered on the page. It is also the `bloxName` unless the `bloxName` attribute is specified.
`bloxName`	Defaults to `id`	The name of the object on the server (the peer).
`dataBlox`		A DataBlox. For example: `dataBlox="<%=myDataBlox %>"`
`dataBloxRef`		The name of a DataBlox already instantiated in the page.
`formElementName`		The name of the rendered page element and the parameter name used for a form POST.
`minimumWidth`	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
`multipleSelect`	`false`	`true` if multiple selections are allowed.
`selectedCube`	The first Cube in the metadata	The Cube object initially selected in the list.
`selectedCubeName`		The name of the cube initially selected in the list.
`size`	`1`	The number of items that are visible in the list. When `multipleSelect` is `true`, `size` has to be greater than 1, or the browser will be default to a size of 4.
`themeClass`		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
`visible`	`true`	`true` if the object is to be rendered in place.

A CubeSelectFormBlox Example

The following example demonstrates how to populate a selection list with all cubes available in a given multidimensional data source.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>

<blox:data id="myDataBlox"
   useAliases="true"
   dataSourceName="Durico"
   connectOnStartup="false"
/>
...
<bloxform:cubeSelect id="cubes"
      dataBloxRef="myDataBlox"
      visible="true" />
...
```

DataSourceSelectFormBlox Reference

This FormBlox adds a selection list of data sources defined to DB2 Alphablox. You can specify the data source type (MDB, RDB, or ALL) or the specific data adapter such as `IBM DB2 JDBC Driver`, `IBM DB2 OLAP Server`, `Hyperion Essbase Adapter`, or `Oracle Driver`.

DataSourceSelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for DataSourceSelectFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
adapterNameFilter	String	The specific type of data sources, based on the adapter, to display in the list. The valid values are the following constants: • DB2Driver • DB2OLAPDeploymentServicesAdapter • DB2OLAPServerAdapter • EssbaseAdapter • EssbaseDeploymentServicesAdapter • JDBC_ODBCBridgeforMSVM • JDBC_ODBCBridgeforSunVM • MSOLAPAdapter • MSSQLDriver • OracleDriver • SybaseDriver • CannedDataAdapter
adminBlox	AdminBlox	An AdminBlox.
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
minimumWidth	String	The minimum width of the element in pixels.
nullDataSourceLabel	String	The first label to display in the selection list which also indicates that a data source has not be selected. Typically, this label is set to instruct users to select a data source. See "A DataSourceSelectFormBlox Example" on page 316.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedDataSource	DataSource	The selected DataSource object (com.alphablox.blox.repository.DataSource)
selectedDataSourceName	String	The names of the selected data source
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

Property	Type	Description
typeFilter	String	The type of data sources to display in the list. Valid values are MDB, RDB, or ALL. The default value is ALL.

The <bloxform:dataSourceSelect> Tag

The following table lists all attributes for the <bloxform:dataSourceSelect> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
adapter		The specific type of data sources, based on the adapter, to display in the list. Valid values are: • IBM DB2 JDBC Driver • IBM DB2 OLAP Server Deployment Services • IBM DB2 OLAP Server • Hyperion Essbase Adapter • Hyperion Essbase Deployment Services • Generic JDBC-ODBC Bridge for MS VM • Generic JDBC-ODBC Bridge for Sun VM • Microsoft SQL Server Driver • Sybase SQL Server Driver • OLEDB for OLAP • Oracle Driver • Canned Data Adapter
adminBloxRef		The AdminBlox already instantiated in the page.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
nullDataSourceLabel		If set, an extra menu item is added to the top and it becomes the default, indicating that a data source has not be selected unless something else is set (via selectedDataSourceName). Typically, this label is set to instruct users to select a data source. See "A DataSourceSelectFormBlox Example" on page 316.
selectedDataSourceName	The first one in the list	The name of the selected data source.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
type	ALL	The type of data sources to display in the list. Valid values are MDB, RDB, or ALL.
visible	true	true if the object is to be rendered in place.

A DataSourceSelectFormBlox Example

The following example creates a drop down selection list containing all multidimensional data sources. The selection list appears with "Select a Data

Source" as the first item. For a complete example that shows how to connect the selection to a DataBlox, see the "Ad Hoc Analysis using DataSourceSelectFormBlox" example under the FormBlox section.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<html>
<head>
   <blox:header />
</head>
<body>
<bloxform:dataSourceSelect id="dataSourceName"
   type="MDB"
   nullDataSourceLabel="Select a Data Source">
</bloxform:dataSourceSelect>
</body>
</html>
```

DimensionSelectFormBlox Reference

This FormBlox adds a selection list of dimensions from the given cube in the given multidimensional data source.

DimensionSelectFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for DimensionSelectFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
`cube`	`Cube`	The Cube object to get the dimensions from.
`cubeName`	`String`	The name of the cube to get the dimensions from.
`dataBlox`	`DataBlox`	The DataBlox from which to get the metadata.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`formValues`	`String[]`	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`minimumWidth`	`String`	The minimum width of the element in pixels.
`multipleSelect`	`boolean`	`true` if multiple selections are allowed. The default is `false`.
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`selectedDimension`	`Dimension`	The selected Dimension object. Defaults to the first one in the list in a single selection list.
`selectedDimensions`	`Dimension[]`	The selected Dimension objects when multiple selections are supported. Defaults to null or an empty array in a multiple selection list.

Property	Type	Description
selectedUniqueName	String	The unique name of the selected dimension. Defaults to the first one in the list in a single selection list.
selectedUniqueNames	String[]	The unique names of the selected dimensions when multiple selections are supported. Defaults to null or an empty array in a multiple selection list.
size	int	The number of lines that are visible in the list. The default is 1. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:dimensionSelect> Tag

The following table lists all attributes for the <bloxform:dimensionSelect> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
cube		The Cube object to get the dimensions from.
cubeName		The name of the cube to get the dimensions from.
dataBlox		The DataBlox from which to get the metadata.
dataBloxRef		The name of a DataBlox already instantiated in the page.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multipleSelect	false	true if multiple selections are allowed.
selectedDimension	Defaults to the first one in the list	The Dimension object initially selected in the list.
selectedDimensionName	Defaults to the first one in the list	The name of the dimension initially selected in the list. For MSAS and SAP BW data sources, the dimension name should be enclosed in square brackets ([ ]).
size	1	The number of items that are visible in the list. When multipleSelect is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

Note: Most FormBlox that create a selection list—CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox—have the same behavior: when the selection list has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this selected*Cube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a size greater than 1 or

when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

DimensionSelectFormBlox Examples

The following example creates a drop down list populated with all dimensions in the specified cube in the specified DataBlox.

```
<bloxform:dimensionSelect id="allDimensions"
   dataBloxRef="dataBlox"
   cubeName="Cube1"
   visible="true" />
```

The following example creates a drop down list populated with all cubes in the specified DataBlox. The dimensions to display in the dimension drop list is determined by the selection of a cube.

```
<table>
<tr>
<td width="100">Select a cube:</td>
<td width="140">Select a dimension:</td>
</tr>
<tr>
<td><bloxform:cubeSelect id="cubes"
      dataBloxRef="dataBlox"
      visible="true" /></td>
<td><bloxform:dimensionSelect id="dimensions"
       dataBloxRef="dataBlox"
       visible="true">
       <bloxform:getChangedProperty formBloxRef="cubes"
          formProperty="selectedCube"
          property="cube"/>
    </bloxform:dimensionSelect></td>
</tr>
</table>
```

EditFormBlox Reference

EditFormBlox adds either a `<text>` or `<textarea>` tag into the rendered page. If the `lines` attribute is not specified or is set to 1, a `<text>` tag is inserted. For better page layout, you may want to add a EditFormBlox inside a table cell in order to put text next to it.

When users click inside the text field (in order to enter information), the input focus is on the EditFormBlox. As soon as users click somewhere else on the page and the input focus is reset, the FormEventListener will call the `valueChanged()` method and the new value is set. Note that hitting the Enter key will not trigger a form POST.

EditFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for EditFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
`charactersPerLine`	`int`	The number of characters allowed per line in the text field. The default `charactersPerLine` is `20`.

Property	Type	Description
focus	boolean	true to set the keyboard focus on this component when it is rendered.
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
lines	int	The number of lines that are rendered in the text field. The default is 1. If this attribute is set to more than 1, a text area is rendered.
maskInput	boolean	true if characters are to be masked (appear as asterisks).
maxCharacters	int	The number of characters allowed in the text field.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.
value	String	The value entered in the text field.

The <bloxform:edit> Tag

The following table lists all attributes for the <bloxform:edit> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
charactersPerLine	20	The number of characters allowed per line in the text field.
focus	true	true to set the keyboard focus on this component when it is rendered.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
lines	1	The number of lines that are rendered in the text field. If this attribute is set to more than 1, a text area is rendered.
maskInput	false	true if characters are to be masked (appear as asterisks).
maxCharacters		The number of characters allowed in the text field.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

An EditFormBlox Example

This example demonstrates how to use an EditFormBlox to allow users to specify the title of a ChartBlox.

- An EditFormBlox is added inside a table cell, with the text to display before it in another cell on the same table row.
- The maxCharacters and charactersPerLine attributes are both set to 30.

- The nested <bloxform:setChangedProperty> tag is used to specify the target object and the property of the object to change.
- As soon as the input focus is set to somewhere else on the page, the ChartBlox's title property is set to the value entered in the text field.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>

<blox:chart id="myChartBlox"
   visible="false"
   width="500"
   height="500">
   <blox:data
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW ('All Products') <CHILD 'All Products'
        <COL ('All Time Periods') <CHILD 'All Time Periods'
         <PAGE(Measures) Sales !" />
</blox:chart>

<html>
<head>
   <blox:header />
</head>

<body>
<table>
<tr>
<td>Title for this chart:</td>
<td>
   <bloxform:edit id="titleEdit"
      charactersPerLine="30"
      maxCharacters="30">
      <bloxform:setChangedPBroperty
         targetRef="myChartBlox"
         targetProperty="title" />
   </bloxform:edit>
</td>
</tr>
</table>
<font size="-1">(When you are done, click anywhere else on the page to set the
title.)</font><p>
<blox:display bloxRef="myChartBlox" />
</body>
</html>
```

MemberSelectFormBlox Reference

This FormBlox adds a selection list of members from a given dimension of a given cube (if necessary) of a given DataBlox. You can set the root members so only the root members and their descendants are displayed in the list. You can also filter the list by specifying whether only members equal to, less than, or greater than a specified generation should be displayed.

MemberSelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for MemberSelectFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
dataBlox	DataBlox	The DataBlox from which to get the metadata.

Property	Type	Description
dimension	Dimension	The Dimension object whose members are to be listed.
dimensionName	String	The unique dimension name whose members are to be listed. For MSAS and SAP BW data sources, the member name should be enclosed in square brackets ([ ]).
filterGeneration	int	The generation to be listed in the list. See the filterOperator property below. The default is 0.
filterOperator	String	The operator to apply to the filterGeneration. Valid values are ==, <, or >.
filterType	int	The operator to apply to the filterGeneration. Valid values are the following constants: • MemberSelectFormBlox.EQUALS • MemberSelectFormBlox.GREATERTHAN • MemberSelectFormBlox.LESSTHAN
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
rootMembers	Member[]	The root Members in this selection list. For MSAS and SAP BW data sources, the member names should be enclosed in square brackets ([ ]).
rootUniqueNames	String[]	The unique names of the root members.
selectedDisplayName	String	The display name of the selected member.
selectedDisplayNames	String[]	The display names of the selected members.
selectedMembers	Member[]	The selected Member objects when multiple selections are supported.
selectedUniqueName	String	The unique name of the selected member.
selectedUniqueNames	String[]	The unique names of the selected members when multiple selections are supported.
size	int	The number of items that are visible in the list.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:memberSelect> Tag

The following table lists all attributes for the <bloxform:memberSelect> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.

Attribute	Default	Description
bloxName	Defaults to id	The name of the object on the server (the peer).
dataBlox		The DataBlox from which to get the metadata.
dataBloxRef		The name of a DataBlox already instantiated in the page.
dimension		The Dimension object whose members are to be listed.
dimensionName		The unique dimension name whose members are to be listed. For MSAS and SAP BW data sources, the member name should be enclosed in square brackets ([ ]).
filterGeneration	0	The generation to be listed in the list. See the filterOperator attribute below.
filterOperator		The operator to apply to the filterGeneration. Valid values are ==, <, or >. The following example lists only members in generation 2: `filterGeneration="2"` `filterOperator="=="`
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multipleSelect	false	true if the list supports multiple selections.
rootMemberName	Defaults to generation 1 roots	The unique name of the root member. For MSAS and SAP BW data sources, the member name should be enclosed in square brackets ([ ]).
rootMemberNames	Defaults to generation 1 roots	The unique names of the root members. For example, `rootMemberNames= "<%=new String[]` `{"[Location].[All Locations]","[Product].[Category]"} %>"`
rootMembers		The root members in this selection list. For example, `rootMembers="<%= mbrs%>"` where mbrs is a Member[] object.
selectedMember	Defaults to the first one in the list in single selection list.	The Member object selected.
selectedMemberName	Defaults to the first one in the list in single selection list	The unique name of the member selected. For MSAS and SAP BW data sources, the member name should be enclosed in square brackets ([ ]).
size	1	The number of items visible in the list. The list is rendered as a drop down list if this value is 1.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

Note: The rootMemberName, rootMemberNames, and selectedMemberName tag attributes all work with unique member names rather than display names. To use the tag attributes when you only have the display name, use the MDBMetaData.resolveMember(*memberName*, true) method to resolve the member name by display name first. See the resolveMember() method of the MDBMetaData interface in the com.alphablox.blox.data.mdb package for more information on how to resolve member names by display name.

Note: Most FormBlox that create a selection list—CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox—have the same behavior: when the selection list

has a `size` of 1 (a drop down list), the first option is automatically set as the initial selection unless this *selectedCube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a `size` greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

A MemberSelectFormBlox Example

See "The getChangedProperty Tag" on page 308 and "MemberSecurityBlox Tags" on page 350.

RadioButtonFormBlox Reference

For each set of radio buttons you add, you can specify whether the buttons in this group should be aligned vertically or horizontally and whether a border should be drawn around the set of buttons. The buttons inside the group are mutually exclusive; selecting one deselects all others in the group. Note that as soon as users select a radio button, the `valueChanged()` method on the FormEventListener is called and the new value is set immediately.

RadioButtonFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for RadioButtonFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
`borderEnabled`	`boolean`	`true` if a border should be drawn around the set of radio buttons The default is `true`.
`buttons`	`String[]`	The values for all the buttons in the set.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The value of the selected radio button.
`formValues`	`String[]`	The value of the selected radio button. In the case of a RadioButtonFormBlox, if more than one value is passed in the array, the first one will be used.
`layout`	Layout	The Layout to apply: HorizontalLayout or VerticalLayout (in com.alphablox.blox.unimodel.core).
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`selectedButton`	`String`	The value for the selected button.
`selectedObject`	`Object`	The selected user Object. If there is no user object associated, then this is the value of the selected button.
`themeClass`	`String`	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:radioButton> Tag

The `<bloxform:radioButton>` tag is used to add a radio button set. To add individual radio buttons, use the `<bloxform:button>` tag. The following table lists all attributes for the `<bloxform:radioButton>` tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
align	horizontal	horizontal or vertical. The default is horizontal.
borderEnabled	true	true if a border should be drawn around the set of radio buttons The default is true.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

The Nested <bloxform:button> Tag

The <bloxform:button> tag is nested inside the <bloxform:radioButton> tag to add buttons.

Attribute	Description
label	The text rendered next to the radio button.
object	The Object value associated with the radio button. This attribute excludes the use of the value attribute.
selected	true if this item is set as the initial selection within the group of radio buttons.
value	The string value associated with the radio button. This attribute excludes the use of the object attribute.

To specify the target object and the property to set when the check box is checked or unchecked, use the nested <bloxform:setChangedProperty> tag. See "The <bloxform:setChangedProperty> Tag Reference" on page 339.

A RadioButtonFormBlox Example

This example demonstrates how to use a RadioButtonFormBlox to allow users to select from two reports. Since each report sets a different query on the DataBlox, selecting one report will deselect the other and reset the data query.

- A RadioButtonFormBlox is added with two buttons.
- The align attribute is set to vertical so the buttons are stacked vertically. The borderEnabled attribute is set to false.
- Notice that no data query is specified in the <blox:data> tag. Since the first radio button Sales By Product is set as the initial selection (selected="true"), its value is used to set the DataBlox's query when the GridBlox is rendered on the page.
- Since we need to set the property on an implicit DataBlox (nested within a GridBlox), we create a session variable ToggleData that we can refer to in the nested <bloxform:setChangedProperty> tag using the following scriptlet:

```
<%
  session.setAttribute("ToggleData",ToggleGridBlox.getDataBlox());
%>
```

- As soon as the user clicks a radio button, the ToggleData's query property is set to the value associated with that button. We then call the updateResultSet() method to update the result set:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>

<blox:grid id="ToggleGridBlox"
   visible="false"
   width="450"
   height="250">
   <blox:data
      dataSourceName="QCC-Essbase"
      useAliases="true" />
</blox:grid>
<%
   session.setAttribute("ToggleData",ToggleGridBlox.getDataBlox());
%>

<html>
<head>
   <blox:header />
</head>
<body>
<b>Display Report:</b>
<bloxform:radioButton id="ReportSelection"
   borderEnabled="false"
   align="vertical" >
   <bloxform:button label="Sales By Product"
      value="<SYM <ROW('All Products') <CHILD 'All Products'
        <COLUMN('All Time Periods') <CHILD 'All Time Periods' Sales !"
      selected="true" />
   <bloxform:button label="Sales By Market"
      value="<SYM <PAGE(Measures) Sales <ROW(\"All Locations\") <CHILD
         \"All Locations\" <COLUMN(\"All Time Periods\") <CHILD
         \"All Time Periods\" !" />
   <bloxform:setChangedProperty
      targetRef="ToggleData"
      targetProperty="query"
      callAfterChange="updateResultSet" />
</bloxform:radioButton>

<blox:display bloxRef="ToggleGridBlox" />

</body>
</html>
```

SelectFormBlox Reference

SelectFormBlox lets you add an HTML <select> element on the page. Just like the HTML form <select> element, you can specify whether multiple selections are allowed and the number of options visible in the list. As soon as users make a selection, the valueChanged() method on the FormEventListener is called and the new value is set immediately.

SelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for SelectFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
items	String[]	An array of all labels in the selection list.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false. When multipleSelect is true, size has to be greater than 1.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedIndexes	int[]	The indices of selected menu items. When the size of the list is 1 and if an empty array is passed, the first item will be selected by default.
selectedItem	String	The label of the selected item.
selectedItems	String[]	The labels of the selected items when multiple selections are supported.
selectedObject	Object	The user Object associated with the first selected item in the list.
selectedObjects	Objects[]	The user Objects associated with the selected items in the list.
size	int	The number of items that are visible in the list. The default is 1. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:select>Tag

The <bloxform:select> tag is used to add a selection list. To add individual options in the list, use the <bloxform:option> tag. The following table lists all attributes for the <bloxform:select> tag:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multipleSelect	false	true if multiple selections are allowed.

Attribute	Default	Description
size	1	The number of options that are visible in the list. When the value is 1, the list is rendered as a drop down list. If none of the options added using the <bloxform:option> tag has selected set to true, the first option on the list is set as the initial selection. When multipleSelect is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place.

The Nested <bloxform:option> Tag

This tag adds an option in the selection list. It has the following attributes:

Attribute	Default	Description
label		The text rendered as an option in the selection list.
object		The Object value associated with the option. This attribute excludes the use of the value attribute.
selected	The first one in the list	true if the item is set as the initial selection in the selection list.
value		The value associated with the option. This attribute excludes the use of the object attribute.

Note: Most FormBlox that create a selection list—CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox—have the same behavior: when the selection list has a size of 1 (a drop down list), the first option is automatically set as the initial selection unless this selected*Cube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a size greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

A SelectFormBlox Example

The following example demonstrates how to use a selection list to allow users to select a chart type.

- A selection list with four options are added to the page. This is a single selection list (multipleSelect is false by default) with all four items visible in the list (size="4").
- The first option is set as the initial selection (selected="true").
- Since we need to set the property of a nested ChartBlox, we create a session variable myChart that can be later referenced in the <bloxform:setChangedProperty> tag:

```
<%
   session.setAttribute("myChart",myPresentBlox.getChartBlox());
%>
```

- Once a selection is made, the chartType property of myChart will be set to the value of the option selected.

The complete code is as follows:

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>
<blox:present id="myPresentBlox"
   visible="false"
   width="560"
   height="450">
   <blox:data
      dataSourceName="QCC-Essbase"
      query="<SYM <ROW('All Products') <CHILD 'All Products'
        <COLUMN('All Time Periods') <CHILD 'All Time Periods' Sales !"
      useAliases="true" />
</blox:present>

<%
   session.setAttribute("myChart",myPresentBlox.getChartBlox());
%>

<html>
<head>
   <blox:header/>
</head>
<body>
<b>Select Chart Type:</b><br>
<bloxform:select id="ChartSelection" size="4">
   <bloxform:option label="Bar" value="Bar" selected="true"/>
   <bloxform:option label="Pie" value="Pie" />
   <bloxform:option label="Line" value="Line" />
   <bloxform:option label="3D Bar" value="3D Bar" />
   <bloxform:setChangedProperty
      targetRef="myChart"
      targetProperty="chartType" />
</bloxform:select>
<blox:display bloxRef="myPresentBlox" />
</body>
</html>
```

TimePeriodSelectFormBlox Reference

TimePeriodSelectFormBlox creates a selection list displaying TimeSeries available in a TimeSchemaBlox. By default, the following TimeSeries entries are displayed:

- Last month
- Last two months
- Last three months
- Last six months
- Last twelve months
- Last quarter
- Last two quarters
- Last four quarters
- Last year
- Last two years
- Month to current
- Quarter to current
- Year to current
- Current month
- Current week

Note that if the time schema does not contain a given period type, then entries that depend on that period type are automatically removed from the defaults. It is also possible to add extra custom entries to the control programmatically. See "TimeSchemaBlox Tag" on page 352.

TimeSeries

The TimeSeries object, as its name suggests, represents a period of time that has the following properties:

- `baseInterval`: Basic period type, such as months, weeks, quarter, and years. It is used to determine the date range.
- `rollups`: Different types of time unit to include in roll-ups. For example, if the TimeSeries is last month, the rollup unit can be month, week, or day.
- `start`: The starting period; the offset from the current time period, with 0 being the current time period; -1, the previous period; -2, the previous 2 periods; 1, the next period, and so on.
- `count`: Number of periods to be included.
- `toDate`: Indicates if this TimeSeries represents a period to date (`TODATE`) or a sequence of periods (`SEQUENCE`). For example, `TODATE(Month)(Week)` indicates month-to-date with Week as the time unit in the rollup. `SEQUENCE(Month,-12,12)(Month,Quarter)` indicates last 12 months with Month and Quarter as the time units in the rollup.

The TimeSeries object is part of the com.alphablox.blox.logic package. Through the TimePeriodSelectFormBlox's nested `<bloxform:timeSeries>` tag, you can specify the time series to be included in the selection list.

TimePeriodSelectFormBlox Properties

When linking FormBlox using the `<bloxform:getChangedProperty>` and `<bloxform:setChangedProperty>` tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for TimePeriodSelectFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
`defaultSeriesVisible`	boolean	`true` if the default time series menu entries should be displayed. The default is `true`. See "TimePeriodSelectFormBlox Reference" on page 329 for a listing of all default entries.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`formValues`	`String[]`	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested `<setChangedProperty>` tag.
`minimumWidth`	`String`	The minimum width of the element in pixels.
`renderHook`	`FormBloxRenderHook`	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
`selectedSeries`	`TimeSeries`	The current selected TimeSeries.

Property	Type	Description
selectedSeriesName	String	The label of the current selected TimeSeries.
selectedSeriesString	String	The current selected time series using the series string. See "The Nested <bloxform:timeSeries> Tag" on page 331 for more information on the time series expression string.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.
timeSchema	TimeSchemaBlox	A TimeSchemaBlox already instantiated in the page.
tuples	Member[][]	The tuples that correspond to the time series.

The <bloxform:timePeriodSelect> Tag

The <bloxform:timePeriodSelect> tag has the following attributes:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
defaultSeriesVisible	true	true if the default time series menu entries should be displayed. See "TimePeriodSelectFormBlox Reference" on page 329 for a listing of all default entries.
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
selectedSeries	The first one in the list	The current selected TimeSeries.
selectedSeriesString	The first one in the list	The current selected time series using the series string. See "The Nested <bloxform:timeSeries> Tag" on page 331 for more information on the time series expression string.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
timeSchemaBloxRef		A TimeSchemaBlox already instantiated in the page.
visible	true	true if the object is to be rendered in place.

The Nested <bloxform:timeSeries> Tag

The <bloxform:timeSeries> tag is nested inside a <bloxform:timePeriodSelect> tag. It has the following attributes:

Attribute	Description
expression	The expression for constructing a TimeSeries. A TimeSeries can be either a SEQUENCE or a TODATE: • For SEQUENCE, specify the period type, the start, the count, and the time unit for rollup. • For TODATE, specify the period type and the time unit for rollup.

For example:

- expression="SEQUENCE(MONTH,-12,12)(MONTH)" indicates last 12 months (starts with twelve months ago and continues for 12 months), with Month as the time unit.
- expression="SEQUENCE(QUARTER,-1,1)(QUARTER)" indicates last quarter (startswith last month and continues for one month), with Quarter as the time unit.
- expression="SEQUENCE(MONTH,-1,1)(WEEK)" indicates last month, with Week as the time unit.
- expression="TODATE(MONTH)(WEEK)" indicates month-to-date with Week as the time unit in the rollup.
- expression="TODATE(QUARTER)(MONTH)" indicates quarter-to-date with Month as the time unit in the rollup.

Valid period types are: Year, Half Year, Quarter, Month, Week, and Day.

name — The displayed label for the TimeSeries.

A TimePeriodSelectFormBlox Example

The following example demonstrates the use of MDBQueryBlox and TimePeriodSelectFormBlox to allow users to select members for the row axis.

- A DataBlox is created first without a query.
- Tuples on the column axis are specified first using the <bloxlogic:tupleList> tag. This TupleList (id="histTuples") will be the TupleList for the column axis. It is defined outside of the <bloxlogic:mdbQuery> tag so it can have an id that we can set the property of this object when a selection of member on the column axis is made.
- A TimePeriodSelectFormBlox is added to show all the default time periods available, with the initial selected member set to the last six months with Month being the rollup unit (selectedSeriesString="SEQUENCE(MONTH,-6,6)(MONTH)").
- A MDBQueryBlox is added with the row and column axes of the query defined. For the row axis, we are displaying only Chocolate Blocks, Chocolate Nuts, and Specialties. For the column axis, histTuples is referenced.
- Once a selection is made by users, the listFromMetadataMembers property of histTuples is updated with the selected members of the TimePeriodSelectFormBlox. The changed() method is called to update the underlying DataBlox.

```
<%@ page import="java.util.Date"%>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<blox:data id="dataBlox" dataSourceName="QCC-MSAS"/>

<bloxlogic:timeSchema id="timeSchema"
   name="QCC-MSAS" dataBloxRef="dataBlox" />
<%
// Since QCC-MSAS only has data up to 2002, we are setting a fixed date
// for today so the example will run. In real applications, you do not need
// to set today's date.
   setToday(timeSchema);
%>
   <bloxlogic:tupleList id="histTuples">
      <bloxlogic:dimension list="<%=timeSchema.getDimensions()%>">
   </bloxlogic:dimension>
</bloxlogic:tupleList>

<bloxform:timePeriodSelect id="historySelector"
   timeSchemaBloxRef="timeSchema"
   selectedSeriesString="SEQUENCE(MONTH,-6,6)(MONTH)"
```

```
   visible="false">
   <bloxform:setChangedProperty formProperty="tuples"
      targetRef="histTuples"
      debugEnabled="true"
      targetProperty="listFromMetadataTuples"
      callAfterChange="changed"/>
</bloxform:timePeriodSelect>

<bloxlogic:mdbQuery id="query" dataBloxRef="dataBlox"
   cubeName="[QCC]">
   <bloxlogic:axis type="rows"
      queryFragment="{[Chocolate Blocks],[Chocolate Nuts],[Specialties]} ON
ROWS " />
   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList tuplesRef="histTuples" />
   </bloxlogic:axis>
</bloxlogic:mdbQuery>

<html>
<head>
   <blox:header />
</head>
<body>

<b>Select a time period: </b>
<blox:display bloxRef="historySelector" />
<blox:grid id="myBlox" width="90%" height="75%"
   toolbarVisible="false" menubarVisible="false">
   <blox:data bloxRef="dataBlox" />
</blox:grid>
</body>
</html>
<%!
   // Set today to a "fixed" date since the sample QCC-MSAS database
   // only has data up to 2002. In real applications, you do not need
   // to set today's date.
   public void setToday(com.alphablox.blox.logic.timeschema.TimeSchemaBlox
timeSchema) throws Exception {
      com.alphablox.blox.logic.timeschema.PeriodType small =
      com.alphablox.blox.logic.timeschema.PeriodType.getSmallest(timeSchema
.getPeriods());
      long end = timeSchema.last(small).getEndDate().getTime();
      timeSchema.setToday(new Date(end));
   }
%>
```

TimeUnitSelectFormBlox Reference

TimePeriodSelectFormBlox creates a selection list displaying time units available in a TimeSchemaBlox. By default, the following time unit entries are displayed:

- Year
- Quarter
- Month
- Week

TimeUnitSelectFormBlox Properties

When linking FormBlox using the <bloxform:getChangedProperty> and <bloxform:setChangedProperty> tags, you may need to specify the name of the property you want to get or change on the target FormBlox. This section lists all properties for TimeUnitSelectFormBlox. For associated methods, see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
formElementName	String	The name of the rendered page element and the parameter name used for a form POST.
formValue	String	The value to return when a selection is made. This is the value to set on the specified property on the target object using the nested <setChangedProperty> tag.
formValues	String[]	The values to return when selections are made. These are the values to set on the specified property on the target object using the nested <setChangedProperty> tag.
items	String[]	An array of all labels in the selection list.
minimumWidth	String	The minimum width of the element in pixels.
multipleSelect	boolean	true if multiple selections are allowed. The default is false.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
selectedPeriodTypes	PeriodType	The current selected PeriodType.
size	int	The number of items that are visible in the list. When the multipleSelect property is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.
timeSchema	TimeSchemaBlox	A TimeSchemaBlox already instantiated in the page.

The <bloxform:timeUnitSelect> Tag

The <bloxform:timeUnitSelect> tag has the following attributes:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
formElementName		The name of the rendered page element and the parameter name used for a form POST.
minimumWidth	The width that fits the longest option in the list	The minimum width for this selection list in pixels. When a selection list contains no options, it appears as a very narrow list. You can use this attribute to make an empty selection list more appealing.
multipleSelect	false	true if multiple selections are allowed.
selectedTimeUnit	The first one in the list	The current selected time unit using the PeriodType string. See "PeriodType" on page 345.
size	1	The number of items that are visible in the list. When multipleSelect is true, size has to be greater than 1, or the browser will be default to a size of 4.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
timeSchemaBloxRef		A TimeSchemaBlox already instantiated in the page.
visible	true	true if the object is to be rendered in place.

Note: Most FormBlox that create a selection list— CubeSelectFormBlox, DimensionSelectFormBlox, MemberSelectFormBlox, SelectFormBlox, and TimeUnitSelectFormBlox— have the same behavior: when the selection list has a `size` of 1 (a drop down list), the first option is automatically set as the initial selection unless this `selected`*Cube/Dimension/Member/Series* attribute is explicitly specified. When the selection list has a `size` greater than 1 or when multiple selections are allowed, at least one option needs to be set as the initial selection or an error may occur.

TreeFormBlox Reference

The TreeFormBlox adds a navigation tree with folders and items based on the tree control in the Blox UI model. For each TreeFormBlox, you can specify whether the items in the tree are draggable, allowing users to move and reorder items, or whether the folder and item labels should be wrapped if the window or frame is too narrow.

Each TreeFormBlox requires one and only one root folder. Depending on your design, you may or may not want to display the root folder by setting the `rootVisible` attribute to `true` or `false`.

For each menu item, you can specify the value for the `href` attribute the same way as you would in an HTML `href` tag attribute. You can also specify the target window if a new page is to be loaded when the item is clicked. There can be links on both the folders and the items. Three tags are needed to create a tree: `<bloxform:tree>`, `<bloxform:folder>`, and `<bloxform:item>`.

TreeFormBlox Properties

This section lists all properties for TreeFormBlox. For associated methods and the inner classes (TreeFormBlox.Folder, TreeFormBlox.Item, TreeFormBlox.ItemDraggedEvent, and TreeFormBlox.ItemDraggedEventListener), see the FormBlox Javadoc documentation under the com.alphablox.blox.form package.

Property	Type	Description
`draggingEnabled`	`boolean`	Specifies if items in the tree can be dragged to other folders or to reorder the items in a folder.
`folderStyle`	`Style`	The Style object used for each folder label in the tree.
`formElementName`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValue`	`String`	The name of the rendered page element and the parameter name used for a form POST.
`formValues`	`String[]`	The names of the rendered page elements and the parameter names used for a form POST. If the case of a RadioButtonFormBlox, if more than one value is passed in the array, the first one will be used.
`itemPositioningEnabled`	boolean	Specifies whether to remain the position of items in a folder. When `draggingEnabled` is set to `true`, and `itemPositioningEnabled` is also set to `true`, users can only drag items into another folder, but cannot reorder the items in the folder.

Property	Type	Description
labelStyle	Style	The Style object used for all item labels in the tree.
renderHook	FormBloxRenderHook	Indicates if the Blox has been rendered; used to provide a custom renderer for a FormBlox.
root	TreeFormBlox.Folder	The root folder of this tree.
rootVisible	boolean	true to show the root folder. The default is true.
selected	String	The name of the selected Item.
selectedItem	TreeFormBlox.Item	The selected Item object.
textWrapped	boolean	true if the labels for all folders and items in this TreeFormBlox should be wrapped if the label is longer than the width of the browser window or frame. The default is true.
themeClass	String	A String containing the theme class name(s) to set on this element. Separate class names with a space.

The <bloxform:tree> Tag

This tag adds a TreeFormBlox on the page. This tag has the following attributes:

Attribute	Default	Description
id		The id of the object that is rendered on the page. It is also the bloxName unless the bloxName attribute is specified.
bloxName	Defaults to id	The name of the object on the server (the peer).
draggingEnabled		Specifies if items in the tree can be dragged to other folders or to reorder the items in a folder.
itemPositoningEnabled		Specifies whether to remain the position of items in a folder. When draggingEnabled is set to true, and itemPositioningEnabled is also set to true, users can only drag items into another folder, but cannot reorder the items in the folder.
rootVisible	true	true to show the root folder. When this attribute is true, the root folder is displayed: When this attribute is false, the root folder is not displayed, with its sub-folders displayed as the top-level folders.
textWrapped	true	true if the labels for all folders and items in this TreeFormBlox should be wrapped if the label is longer than the width of the browser window or frame.
themeClass		The name(s) of theme class(es) to set on this element. If more than one classes are specified, separate the names with a space.
visible	true	true if the object is to be rendered in place. The default is true.

To add a folder in the tree, use the nested <bloxform:folder> tag.

The Nested <bloxform:folder> Tag

Add this tag inside the <bloxform:tree> tag to add a folder. Note that each tree needs one and only root folder. Therefore, typically you should have at least two levels of folders in your tree. See "A TreeFormBlox Example" on page 338 for details.

To add items in the folder, use the nested <bloxform:item> tag. The <bloxform:folder> tag has the following attributes:

Attribute	Description
draggable	Specifies whether this folder should be draggable.
expanded	Specifies whether the folder should be expanded when it is rendered. The default is false.
href	The URI to load when the folder is clicked. This can be a link or a JavaScript function.
imageURL	The URL of the custom image to use. If themeBasedImage is set to true, your custom images need to reside in the directory where the theme's images are stored in the DB2 Alphablox repository. For details on how to specify the URL, see the discussion for the same property and tag attribute in "Nested <bloxui:menuItem> Tag Attributes" on page 395.
label	The text rendered next to the folder.
name	The name for the folder object.
object	The user object associated with this folder.
target	The target window or frame to load the URI specified in href. By default, the URI specified is loaded in the same window or frame.
themeBasedImage	Set to true to use theme-based images. Theme-based images need to reside in the directory where each theme's images are stored in the repository.
tooltip	The text displayed when the mouse hovers over the folder.

The Nested <bloxform:item> Tag

This tag adds individual menu items in folders. It has the following attributes:

Attribute	Description
draggable	Specifies whether this item should be draggable.
href	The URI to load when the item is clicked. This can be a link or a JavaScript function.
imageURL	The URL of the custom image to use. If themeBasedImage is set to true, your custom images need to reside in the directory where the theme's images are stored in the DB2 Alphablox repository. For details on how to specify the URL, see the

	discussion for the same property and tag attribute in "Nested <bloxui:menuItem> Tag Attributes" on page 395.
`label`	The text rendered next to the item.
`name`	The name for the item object.
`object`	The user object associated with this item.
`target`	The target window or frame to load the URI specified in `href`. By default, the URI specified is loaded in the same window or frame.
`themeBasedImage`	Set to `true` to use theme-based images. Theme-based images need to reside in the directory where each theme's images are stored in the repository.
`tooltip`	The text displayed when the mouse hovers over the item.

A TreeFormBlox Example

The following example creates a non-draggable menu tree with two folders. The root folder is not visible. It assumes the menu tree is in one frame, and when users click a menu item, a new page is loaded into a different target frame.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxformtld" prefix="bloxform" %>

<html>
<head>
    <blox:header/>
</head>
<body>
<bloxform:tree id="myMenu" rootVisible="false" >
   <bloxform:folder> <%--root folder--%>
      <bloxform:folder label="Sales Analysis">
         <bloxform:item label="Sales Trend by Region"
            href="salesByRegion.jsp"
            target="mainFrame" />
         <bloxform:item label="Sales by Store"
            href="salesByStore.jsp"
            target="mainFrame" />
         <bloxform:item label="Units Sold by Product"
            href="unitsSoldByProduct.jsp"
            target="mainFrame" />
      </bloxform:folder>

      <bloxform:folder label="Variance Analysis" expanded="false">
         <bloxform:item label="Sales Variance"
            href="varianceSales.jsp"
            target="mainFrame" />
         <%--Pop up an alert window as the report is not available.--%>
         <bloxform:item label="Ad-Hoc Variance Analysis"
            href="javascript:alert(\"Currently unavailable.\")" />
      </bloxform:folder>
   </bloxform:folder>
</bloxform:tree>
</body>
</html>
```

Note: You should use escaped double quotes inside the JavaScript call for the `href` attribute. Single quotes will cause JavaScript errors.

The <bloxform:getChangedProperty> Tag Reference

This tag is used to link FormBlox, allowing one FormBlox to get the selected property value of another FormBlox.

Attribute	Description
`debugEnabled`	When set to `true`, this turns on debug logging of property changes.
`formBlox`	The source FormBlox. This is the FormBlox where the property value is from. For example, `<bloxform:select id="mySelectFormBlox" ...>` `  <bloxform:getChangedProperty` `  formBlox="<%= anotherFormBloxn%>"` `  ..../>` `</bloxform:select>` Either this attribute or the `formBloxRef` attribute should be specified.
`formBloxRef`	The name of the source FormBlox already instantiated in the page. This is the FormBlox where the value is from. Either this attribute or the `formBlox` attribute should be specified.
`formProperty`	The name of the property on the source FormBlox whose change causes a notification.
`property`	The name of the property on the target FormBlox.

For examples of this tag, see "The getChangedProperty Tag" on page 308.

The <bloxform:setChangedProperty> Tag Reference

This tag is used to link FormBlox, allowing the selection in one FormBlox to set the selected property value of another FormBlox. It can set the properties on any Java bean since it is using the normal Java bean introspection. See the discussion of "The FormPropertyLink Object" on page 308.

Attribute	Description
`callAfterChange`	The method to call after the property is changed on the target. This method cannot have parameters. A common scenario is when a DataBlox property is changed, the `updateResultSet()` method needs to be called following the property change. Another example is when the property of a TupleList, a CrossJoin, or an Axis object is updated, its `changed()` method should be called in order to notify the query that the value has changed.
`debugEnabled`	When set to true, this turns on debug logging of property changes.
`formProperty`	The name of the property to propagate on changes.
`target`	The target object. It can be any Java bean. This is the target whose property will be changed. Either this attribute or the `targetRef` attribute should be set.

`targetRef`	The name of a bean already instantiated in the page. This is the target whose property will be changed. Either this attribute or the `target` attribute should be set.
`targetProperty`	The name of the property to change on the target bean.

For examples of this tag, see:

- "A CheckBoxFormBlox Example" on page 312
- "An EditFormBlox Example" on page 320
- "A RadioButtonFormBlox Example" on page 325
- "A SelectFormBlox Example" on page 328
- "A TimePeriodSelectFormBlox Example" on page 332
- "MemberSecurityBlox Tags" on page 350

Chapter 21. Business Logic Blox and TimeSchema DTD Reference

This chapter contains reference material for the three business logic Blox—TimeSchemaBlox, MDBQueryBlox, and MemberSecurityBlox. The Data Type Definition (DTD) for creating a TimeSchema XML is also described.

Blox Logic Tags Overview

DB2 Alphablox provides three business logic Blox to help you add commonly needed business logic in analytic applications— TimeSchemaBlox, MDBQueryBlox, and MemberSecurityBlox: These business logic Blox and FormBlox (discussed in Chapter 20, "Blox Form Tag Reference," on page 305) are designed to solve two commonly encountered problems during analytical application development: the need for data-aware business logic and the need to maintain state.

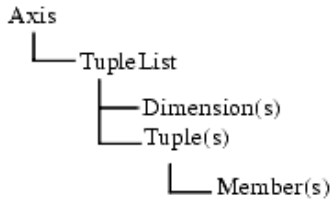
- All three Blox support IBM DB2 OLAP Server, Essbase, Microsoft Analysis Service, and SAP BW data sources.
- These Blox and their associated objects reside in the com.alphablox.blox.logic and com.alphablox.blox.logic.timeschema packages.
- The tags for these business logic Blox are provided in the Blox Logic Tag Library. To use the Blox Logic tags, you need to include the following taglib import statement in your page:

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic" %>
```

MDBQueryBlox

MDBQueryBlox is an object representation of a multidimensional data query. It allows you to manipulate an MDB query without using the query language associated with the data source. Using the `<bloxlogic:mdbQuery>` tag or its API, you can manipulate parts of the query such as changing parts of the tuples of an axis. Once a change is made in MDBQueryBlox (by calling its `changed()` method), its source DataBlox is automatically updated and the data query re-executed.

An MDBQueryBlox has three axes: rows, columns and the slicer (or the "page" axis). Each represents a particular part of the query. Each axis is an Axis object composed of multiple tuples, and therefore a TupleList. Each TupleList is defined by dimensions and Tuples.



A Tuple contains a list of members that can be from one or multiple dimensions. The following example shows a GridBlox with one tuple on the row axis, consisting of member "All Products" (the root member) of the "All Products" dimension, and two tuples on the column axis, consisting of two members from the same dimension.

All Products	Qtr 1 01	Qtr 2 01
All Products	1770633.39	2826604.715

The following example shows a GridBlox with two tuples for the column axis. Each tuple is formed by members from two dimensions—All Time Periods and Scenario.

	Qtr 1 01	Qtr 2 01
All Products	Actual	Actual
All Products	1770633.39	2826604.715

An Axis object can also be composed of one or more CrossJoin objects. A CrossJoin produces a "cross product" of the tuples that it joins. For example, if `tuples1 = {"Jan", "Feb"}` and `tuples2 = {"Colas", "Root Beer"}`, then `CrossJoin.getTuples()` will return `{{"Jan", "Colas"}, {"Jan", "Root Beer"}, {"Feb", "Colas"},{"Feb", "Root Beer"}}`. The following example shows four tuples on the column axis as a result of a cross join of:

- "Qtr 1 01" and "Qtr 2 01" from the "All Time Periods" dimension
- "Actual" and "Forecast" from the "Scenario" dimension

	Qtr 1 01		Qtr 2 01	
All Products	Actual	**Forecast**	Actual	**Forecast**
All Products	1770633.39	1780642.53	2826604.715	2809041.365

A TupleList represents a set of tuples that may be part of an Axis or a CrossJoin. Tags for MDBQueryBlox generally have the following nested relationship:

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:tupleList>
```

Or

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:crossJoin>
         <bloxlogic:tupleList>
```

By setting the dimensions and members for the row, column, or page axis in MDBQueryBlox, you can make changes to the query in parts. For example, using

MemberSelectFormBlox, you can create a member selection list that allows users to select members of interest to display on the row axis. The selected members can then be used to set the values of the TupleList's `listFromMetaDataMembers` or `ListFromMetadataTuples` property. This updates the DataBlox referenced once the `TupleList.changed()` method is called. For an example, see "An MDBQueryBlox Example" on page 349.

Specifying TupleList for Each Axis

To define a data query using MDBQueryBlox, specify the type of the axis (`rows`, `columns`, or `pages`) and then the TupleList(s) that form the axis.

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<bloxlogic:mdbQuery id="query" dataBloxRef="myDataBlox">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>

       ... Define the dimension(s) and tuple(s)

      </bloxlogic:tupleList>
   </bloxlogic:axis>

...Specify the list of tuples for rows and columns in the same way

</bloxlogic:mdbQuery>
```

Example 1: A Simple Query: This example demonstrates how to define a simple query with one tuple each on the row and column axes. Each tuple consists of the root member from a dimension. The rendered GridBlox looks as follows:

All Products	All Time Periods
All Products	14072286.895

The tags that specify this query is as follows:

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<bloxlogic:mdbQuery id="myQuery" dataBloxRef="someDataBlox">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Products</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>All Products</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>

   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Time Periods</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>All Time Periods</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

Example 2: Two Dimensions on an Axis: This example demonstrates how to define a query with a tuple formed by members from different dimensions. The following GridBlox has:

- Chocolate Blocks and Chocolate Nuts from the All Products dimension on the row axis
- A tuple formed by Qtr 1 01 from All Time Periods and Actual from Scenario on the column axis

- Another tuple formed by Qtr 2 01 from All Time Periods and Actual from Scenario on the column axis

	Qtr 1 01	Qtr 2 01
All Products	Actual	Actual
Chocolate Blocks	347784.61	428594.33
Chocolate Nuts	660425.345	1294959.57

The tags that specify the members on the row and column axes are as follows:

```
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<bloxlogic:mdbQuery>
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Products</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>Chocolate Blocks</bloxlogic:member>
         </bloxlogic:tuple>
         <bloxlogic:tuple>
            <bloxlogic:member>Chocolate Nuts</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
   <bloxlogic:axis type="columns">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>All Time Periods</bloxlogic:dimension>
         <bloxlogic:dimension>Scenario</bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>Qtr 1 01</bloxlogic:member>
            <bloxlogic:member>Actual</bloxlogic:member>
         </bloxlogic:tuple>
         <bloxlogic:tuple>
            <bloxlogic:member>Qtr 2 01</bloxlogic:member>
            <bloxlogic:member>Actual</bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

MemberSecurityBlox

MemberSecurityBlox provides a list of members a user has access to on a given dimension. It constructs the list by performing a suppressNoAccess on the DataBlox based on the specified MemberSecurityFilter. To set a MemberSecurityFilter, specify the dimension and the member(s) in that dimension using its `addMemeber()` or `setMember()` method.

TimeSchemaBlox

TimeSchemaBlox builds a time table for a given data source based on your definition of a TimeSchema. Using the TimeSchema Data Type Definition (DTD), you can define how the Time dimension is structured by specifying:

- Name(s) of the time dimension(s)
- The generation levels for Year, Quarter, Month and Week
- Start date of the time period in the cube
- Whether Normal Calendar time/Weekly time should be applied
- If the length of a year is exceptional (such as 48-week year)

The XML file containing the definition of the TimeSchema should be named `timeschema.xml` and stored in your application's `WEB-INF/` directory. A `timeschema.dtd` file should also be stored in the same location. The Data Type Definition (DTD) used to define the TimeSchema XML is described in "TimeSchema XML DTD" on page 353.

Once the time schema is configured, TimeSchemaBlox and its related objects will take care of determining the set of members mapped to a given date or time period. The time schema can perform basic date arithmetic and has the ability to produce a sequence of members between dates. Through TimeSchemaBlox, the time schema is available to TimePeriodSelectFormBlox and TimeUnitSelectFormBlox to create a selection list for users to choose a desired time period and unit. Or you can find out information such as the names of the time dimension(s), the current month, quarter, year, or the previous two months, quarters, years, and more through the TimeSchemaBlox API. For TimePeriodSelectFormBlox and TimeUnitSelectFormBlox, see Chapter 20, "Blox Form Tag Reference," on page 305.

The TimeSchemaManager object in the com.alphablox.blox.logic.timeschema package is the global manager that provides access to the TimeSchema object. You can get to the TimeSchema using the TimeSchemaManager's `getTimeSchema()` method. More conveniently, the `<bloxlogic:timeSchema>` tag does the work for you.

PeriodType

PeriodType describes the period type for a TimeSeries. Valid period types are the following constants:

- `PeriodType.YEAR`
- `PeriodType.HALFYEAR`
- `PeriodType.QUARTER`
- `PeriodType.MONTH`
- `PeriodType.WEEK`
- `PeriodType.DAY`.

See the discussion of TimeSeries in the "TimePeriodSelectFormBlox Reference" on page 329 for more details.

TimeMember

TimeMember is an interface representing a slice of the TimeSchema. With TimeMember, you can find out when this slice of the time table begins, when it ends, what tuple is associated with the date, or what Member objects are associated with the slice.

TimeSeries

TimeSeries represents a series of periods with the following properties:

- `baseInterval`: Basic period type, such as `month`, `week`, `quarter`, and `year`. It is used to determine the date range.
- `rollups`: Different types of time unit to include in roll-ups.
- `start`: The starting period; the offset from the current time period, with 0 being the current time period; -1, the previous period; -2, the previous 2 periods; 1, the next period, and so on.
- `count`: Number of periods to be included.

- `toDate`: Indicates if this TimeSeries represents a period to date (`TODATE`) or a sequence of periods (`SEQUENCE`). For example, `TODATE(Month)(Week)` indicates month-to-date with Week as the time unit in the rollup. `SEQUENCE(Month,-12,12)(Month,Quarter)` indicates last 12 months with Month and Quarter as the time units in the rollup.

A time series can be expressed as a string such as `SEQUENCE(QUARTER, 0, 1)(WEEK)`, which means this is a sequence of quarters, starting from this quarter (0), for a count of 1 quarter, and the unit for roll-ups is Week. With a defined time schema, you can use the TimeSeries bean to construct a time series. The following example shows how a TimeSeries of last quarter with Month as the unit for roll-ups may be constructed:

```
<%
TimeSeries lastQuarter = TimeSeries.parseString("SEQUENCE(Quarter, -1, 1)
(MONTH)");
%>
```

You can also specify a time series when using the TimePeriodSelectFormBlox to create a selection list out of the specified periods. For more information, see "TimePeriodSelectFormBlox Reference" on page 329.

MDBQueryBlox Tags

This section describes the tag syntax for MDBQueryBlox:

<bloxlogic:mdbQuery> Tag Attributes

The `<bloxlogic:mdbQuery>` tag has the following attributes:

Attribute	Description
`id`	The unique `id` of this MDBQueryBlox.
`dataBloxRef`	A DataBlox that is already instantiated in the page.
`cubeName`	the name of the cube in the given DataBlox.

General Tag Syntax

Tags related to MDBQueryBlox have the following nested relationship:

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:tupleList>
```

Or

```
<bloxlogic:mdbQuery>
   <bloxlogic:axis>
      <bloxlogic:crossjoin>
         <bloxlogic:tupleList>
```

The <bloxlogic:tupleList> tag can also stand alone outside the <bloxlogic:mdbQuery> tag. See "<bloxlogic:tupleList> Tag" on page 348 for more details.

Nested <bloxlogic:axis> Tag

This tag needs to be nested inside a <bloxlogic:mdbQuery> tag. It has the following attributes:

Attribute	Description
mutable	true if the axis will change in response to changes in its associated DataBlox. The default is false. If user interaction is allowed in your presentation Blox, you should set mutable to true on both your row and column axes so the changes caused by user's data navigation actions will be reflected correctly. Sometimes you may not need to set mutable to true if you do not allow data drilling (such as by removing the Toolbar and the menu bar and setting the component's clickable attribute to false), and only want the presentation Blox to display data based on some predefined selections.
queryFragment	The query fragment that represents the axis.
type	The type of the axis. Valid values are rows, columns, or pages.

Nested <bloxlogic:crossJoin> Tag

This is a nested tag inside the <bloxlogic:axis> tag. It has no attribute. Inside the <bloxlogic:crossJoin> tag, specify the TupleLists that should be joined. See the example below.

An CrossJoin Example

The following example demonstrates how two TupleLists, one from [Time].[Calendar] and another from [Scenario].[All Scenario].[Actual], are joined on the column axis.

```
<bloxlogic:timeSchema id="timeSchema"
   name="QCC-MSAS" dataBloxRef="myDataBlox" />

<bloxlogic:mdbQuery>
   <bloxlogic:axis type="columns" mutable="true">
      <bloxlogic:crossJoin>

         <bloxlogic:tupleList>
            <bloxlogic:dimension>
               [Time.Calendar]
            </bloxlogic:dimension>
            <bloxlogic:tuple>
               <bloxlogic:member>
                  [Time.Calendar].[2000]
               </bloxlogic:member>
            </bloxlogic:tuple>
         </bloxlogic:tupleList>

         <bloxlogic:tupleList>
            <bloxlogic:dimension>
               [Scenario]
            </bloxlogic:dimension>
            <bloxlogic:tuple>
```

```
            <bloxlogic:member>
               [Scenario].[All Scenario].[Actual]
            </bloxlogic:member>
         </bloxlogic:tuple>
      </bloxlogic:tupleList>

   </bloxlogic:crossJoin>
 </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

<bloxlogic:tupleList> Tag

The <bloxlogic:tupleList> tag is a nested tag within the <bloxlogic:axis> tag. It can also stand alone without being nested. This allows you to specify the id of the TupleList to be referenced later such as in a <bloxform:setChangeProperty> tag.

Attribute	Description
id	The id of the object.
tuplesRef	A TupleList object already instantiated in the page.

The <bloxlogic:tupleList> tag has two nested tags:

- "Nested <bloxlogic:dimension> Tag" on page 348
- "Nested <bloxlogic:tuple> Tag" on page 348

Nested <bloxlogic:dimension> Tag

To specify the dimension in an axis, name the dimension within the <bloxlogic:dimension> tag. For example:

```
<bloxlogic:dimension>[Scenario]</bloxlogic:dimension>
```

To specify a list of dimensions, use the following attribute:

Attribute	Description
list	An array of dimension names. A useful scenario is when you need to dynamically get the list of dimensions, such as from the TimeSchema: `list="<%=myTimeSchema.getDimensions()%>"` This gives you a list of TimeSchema dimensions defined in the application's timeshema.xml file, assuming a TimeSchemaBlox with an id of myTimeSchema is already created. You can also construct the list as follows: `list="<%= new String[] {"dim1", "dime2"} %>"`

Nested <bloxlogic:tuple> Tag

You can have multiple <bloxlogic:tuple> tags inside a <bloxlogic:tupleList> tag. Each <bloxlogic:tuple> tag can have one or more nested <bloxlogic:member> tags. To specify a list of tuples, use the following attribute:

Attribute	Description
list	An array of string array or Member array.

Nested <bloxlogic:member> Tag

This tag has no attributes. Members available to the dimensions and tuple specified should be added between the opening and closing tag. For example:

```
<bloxlogic:tuple>
   <bloxlogic:member>
      [Locations].[All Locations]
   </bloxlogic:member>
   <bloxlogic:member>
      [Products].[All Products]
   </bloxlogic:member>
</bloxlogic:tuple>
```

An MDBQueryBlox Example

The following example demonstrates the use of MDBQueryBlox and MemberSelectFormBlox to allow users to select members for the row axis.

- A DataBlox is created first without a query.
- Tuples on the row are specified first using the `<bloxlogic:tupleList>` tag. This TupleList (`id="rowTuples"`) will be the TupleList for the row axis. It is defined outside of the `<bloxlogic:mdbQuery>` tag so it can have an id that we can set the property of this object when a selection of member on the row axis is made. This TupleList will get all members under [Locations].[All Locations]. Notice that unique member names are required.
- A MDBQueryBlox is added with the row and column axes of the query defined. For the column axis, we are displaying only Sales, COGS, Gross Margin in the Measures dimension. For the row axis, `rowTuples` is referenced.
- A MemberSelectFormBlox is added to show the members under [Locations], with the initial selected member set to [Locations].[All Locations]. Notice that this setting is the same as the setting in `rowTuples`.
- Once a selection is made by users, the `listFromMetadataMembers` property of `rowTuples` is update. The `changed()` method is called to update the underlying DataBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>
<html>
<head>
   <blox:header />
</html>
<body>
<blox:data id="myDataBlox"
   dataSourceName="QCC-MSAS"/>

<bloxlogic:tupleList id="rowTuples">
   <bloxlogic:dimension>[Locations]</bloxlogic:dimension>
   <bloxlogic:tuple>
      <bloxlogic:member>
         [Locations].[All Locations]
      </bloxlogic:member>
   </bloxlogic:tuple>
</bloxlogic:tupleList>

<bloxlogic:mdbQuery id="myQuery" dataBloxRef="myDataBlox" cubeName="[QCC]">
   <bloxlogic:axis type="rows">
      <bloxlogic:tupleList tuplesRef="rowTuples" />
   </bloxlogic:axis>
   <bloxlogic:axis type="columns" mutable="true">
      <bloxlogic:tupleList>
         <bloxlogic:dimension>[Measures]</bloxlogic:dimension>
            <bloxlogic:tuple>
```

```
            <bloxlogic:member>[Measures].[Sales]</bloxlogic:member>
         </bloxlogic:tuple>

         <bloxlogic:tuple>
            <bloxlogic:member>[Measures].[COGS]</bloxlogic:member>
         </bloxlogic:tuple>

         <bloxlogic:tuple>
            <bloxlogic:member>[Measures].[Gross Margin %]
         </bloxlogic:member>
      </bloxlogic:tuple>
   </bloxlogic:tupleList>
</bloxlogic:axis>
</bloxlogic:mdbQuery>

<bloxform:memberSelect id="locationSelector"
   dataBloxRef="myDataBlox"
   dimensionName="[Locations]"
   selectedMemberName="[Locations].[All Locations]"
   multipleSelect="true" visible="false">
   <bloxform:setChangedProperty formProperty="selectedMembers"
      targetRef="rowTuples"
      targetProperty="listFromMetadataMembers"
      callAfterChange="changed"/>
</bloxform:memberSelect>

<b>Select Locations for Row Axis:</b>
<blox:display bloxRef="locationSelector" />
<blox:grid id="myGridBlox" width="100%" height="100%">
   <blox:data bloxRef="myDataBlox" />
</blox:grid>
</body>
</html>
```

MemberSecurityBlox Tags

This section describes the tag syntax for MemberSecurityBlox. For its methods, see the MemberSecurityBlox class in com.alphablox.blox.logic package in the Javadoc documentation.

<bloxlogic:memberSecurity>

The <bloxlogic:memberSecurity> tag has the following attributes:

Attribute	Description
id	The unique id of this MemberSecurityBlox.
cubeName	The name of the cube to set suppressNoAccess on.
dataBlox	A DataBlox.
dataBloxRef	The name of a DataBlox already instantiated on the page.
dimensionName	The name of the dimension to set suppressNoAccess on.

<bloxlogic:memberSecurityFilter>

The <bloxlogic:memberSecurityFilter> tag has the following attributes:

Attribute	Description
dimensionName	The name of the dimension.
memberName	The name of a member.

A MemberSecurityBlox Example

The following example demonstrates the use of the tags and their nested relationship:

```
<bloxlogic:memberSecurity id="memberSecurity"
   dataBloxRef="dataBlox"
   dimensionName="Market">
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Profit" />
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Inventory" />
</bloxlogic:memberSecurity>
```

Assuming we want to provide a report on Profit and Inventory by Market, and we want:

- A selection list populated with members from the Market dimension to allow the user to select the markets of interest.
- Only members in the Market dimension for which the user can access the data on Profit and Inventory to be on the list.

To do so, we need to:

- Set the MemberSecurityBlox's `dimensionName` attribute to Market.
- Set the member security filters to Profit and Inventory in the Measures dimension.

The complete code is as follows:

```
<%@ page import="com.alphablox.blox.logic.MemberSecurityFilter"%>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxlogictld" prefix="bloxlogic"%>

<html>
<head>
   <blox:header />
</head>

<blox:data id="dataBlox" query="!"
   dataSourceName="essbaseFilter"/>

<bloxlogic:memberSecurity id="memberSecurity"
   dataBloxRef="dataBlox"
   dimensionName="Market">
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Profit" />
   <bloxlogic:memberSecurityFilter
      dimensionName="Measures"
      memberName="Inventory" />
</bloxlogic:memberSecurity>

<bloxform:select id="members"
   visible="false"
   multipleSelect="true"
   size="5" >
   <%
      members.setItems(memberSecurity.getDisplayMemberNames());
   %>
</bloxform:select>
```

```
<body>
   <blox:display bloxRef="members" />
</body>
</html>
```

If the database administrator has limited access for the logged in user as follows:

Market	Profit	Inventory
New York	#No Access	8723
Massachusetts	#No Access	3699
Florida	#No Access	5632
Connecticut	#No Access	4852
New Hampshire	#No Access	2838
East	#No Access	25744
California	#No Access	#No Access
Oregon	#No Access	#No Access
Washington	#No Access	#No Access
Utah	#No Access	#No Access
Nevada	#No Access	#No Access
West	29861	#No Access
Texas	#No Access	#No Access
Oklahoma	#No Access	#No Access
Louisiana	#No Access	#No Access
New Mexico	#No Access	#No Access
South	#No Access	#No Access
Illinois	#No Access	#No Access
Ohio	#No Access	#No Access
Wisconsin	#No Access	#No Access
Missouri	#No Access	#No Access
Iowa	#No Access	#No Access
Colorado	#No Access	#No Access
Central	#No Access	#No Access
Market	#No Access	#No Access

The following members will be returned as a result:

- New York
- Massachusetts
- Florida
- Connecticut
- New Hampshire
- East
- West

TimeSchemaBlox Tag

This section describes the tag syntax for TimeSchemaBlox. For related methods, see the com.alphablox.blox.logic.timescheme package in the Javadoc documentation.

<bloxlogic:timeSchema>

The `<bloxlogic:timeSchema>` tag has the following attributes:

Attribute	Description
`id`	An unique identifier for this TimeSchemaBlox.
`dataBloxRef`	The name of a DataBlox already instantiated in the page.
`name`	The name of the TimeSchemaBlox. The name

	should match the name specified in the `timeschema.xml` file (the `name` attribute of the `<timeSchema>` element).
`today`	The current date. The date should be specified in the format of `"mm/dd/yyyy"`. For example: today = ″05/01/2003″

A TimeSchemaBlox Example

The `<bloxlogic:timeSchema>` tag creates a TimeSchemaBlox that can be referenced by a TimePeriodSelectFormBlox, a TimeUnitSelectFormBlox, or a MDBQueryBlox to create a time period selection list or to manipulate the data query. The following code snippet shows a TimeSchemaBlox used by a TimePeriodSelectFormBlox . By default, TimePeriodSelectFormBlox presents the users with a list of time periods to choose from. When a selection is made, the `histTuples`' `listFromMetadataTuples` property is changed accordingly as the `changed()` method is called. For a complete example, see "A TimePeriodSelectFormBlox Example" on page 332.

```
<blox:data id="dataBlox" dataSourceName="MSAS" />
<bloxlogic:timeSchema id="timeSchema" name="MSAS"
   dataBloxRef="dataBlox" />
<bloxlogic:tupleList id="histTuples">
   <bloxlogic:dimension list="<%=timeSchema.getDimensions()%>">
   </bloxlogic:dimension>
</bloxlogic:tupleList>
<bloxform:timePeriodSelect id="historySelector"
   timeSchemaBloxRef="timeSchema"
   selectedSeriesString="SEQUENCE(QUARTER,-1,1)(QUARTER)"
   visible="false">
   <bloxform:setChangedProperty formProperty="tuples"
      targetRef="histTuples"
      targetProperty="listFromMetadataTuples"
      callAfterChange="changed"/>
   </bloxform:timePeriodSelect>
```

TimeSchema XML DTD

The definition of a TimeSchema should be stored in the `timeschema.xml` file in a web application's `WEB-INF/` directory. This file is reloaded each time it changes. The best way to create your `timeschema.xml` file is to copy the one in the FastForward application and then make changes to it. The FastForward application directory is located at:

```
<alphablox_dir>/system/ApplicationStudio/FastForward/
```

Each `timeschema.xml` file can contain multiple TimeSchema, one for each of the data sources needed for your application. This section contains the following topics:

Structure of timeschema.xml

This file has the following general structure:

```
<timeSchemas>
   <timeSchema dataSource="QCC-MSAS" name="QCC-MSAS" type="Weekly1D"
cube="qcc">
      ...
   </timeSchema>

   <timeSchema dataSource="TBC" name="tbc" type="Normal1D">
      ...
   </timeSchema>
</timeSchemas>
```

- timeSchemas is the outmost element.
- Use the timeSchema element for each data source needed in your application.

The following are two examples to demonstrate the general structure.

A Sample TimeSchema for IBM DB2 OLAP Server or Hyperion Essbase Data Sources

The following is an example using an IBM DB2 OLAP Server or Hyperion Essbase data source:

```
<timeSchema dataSource="TBC" name="tbc" type="Normal1D">
   <calculation startDate="01/01/1998"/>
   <dimension name="Year" rootMember="Year">
      <level type="years" generation="1" match="Year"/>
      <level type="quarters" generation="2" match="Qtr{0}"/>
      <level type="months" generation="3" match="{MMM}"/>
   </dimension>
</timeSchema>
```

- This TimeSchema is associated with a data source called TBC.
- This TimesShema's name is tbc. This is the name used to look up a TimeSchema
- This TimeSchema is of type "Normal1D". This type parameter indicates how the length of a year is calculated (Normal or Weekly), and whether the year members are in the same dimension (1D) as the rest of the calendar related members. In this case the year is the same as the "Normal" calendar year and the year members are in same dimension as the rest of the members.
- The <calculation> element in the entry specifies that the time table should start with January 1, 1998.
- The <dimension> element in the entry specifies that the members are located in the Year dimension, and that the root member is Year.
- Within the <dimension> element are three <level> elements:
  - The "years" level is found on generation 1 of the Year dimension and its members should match the pattern "Year"
  - The "quarters" level is found on generation 2 of the Year dimension and its members should match the pattern "Qtr{0}" (such as Qtr1 and Qtr2).
  - The "months" level is found on generation 3 of the Year dimension and its members should match the pattern "{MMM}" (localized three-character month abbreviation such as Jan, Feb, and Mar.)

A Sample TimeSchema for Microsoft Analysis Services Data Sources

The following is an example using a Microsoft Analysis Services data source. This entry is for the QCC-MSAS data source that ships with DB2 Alphablox.

```
<timeSchema dataSource="QCC-MSAS"
   name="QCC-MSAS"
   type="Weekly1D"
```

```
   cube="qcc">
   <calculation startDate="01/30/2000">
      <exceptionYear lengthWeeks="48">2000</exceptionYear>
   </calculation>
   <dimension name="[Time].[Fiscal]">
      <level type="years" generation="2" match="[Time].[Fiscal].[All
       Time Periods].[FY{0000}]"/>
      <level type="quarters" generation="3"
       match="[Time].[Fiscal].[All Time Periods].[FY{0000}].[Qtr {0}
       FY{00}]"/>
      <level type="months" generation="4"
           match="[Time].[Fiscal].[All Time Periods].[FY{0000}].
       [Qtr {0} FY{00}].[{MMM} FY{00}]"/>
      <level type="weeks" generation="5" match="[Time].[Fiscal].[All
       Time Periods].[FY{0000}].[Qtr {0} FY{00}].[{MMM} FY{00}].[{00}-
       {00}-{0000}]"/>
   </dimension>
</timeSchema>
```

- The cube attribute is necessary in this case since MSAS data sources can have multiple cubes.
- The type attribute is set to Weekly1D as this is a year with only 48 weeks. The <exceptionYear> element's lengthWeeks attribute is set to 48 and the time table should be built starting from January 30, 2000.
- The <level> element has a match attribute. Since the TimeSchema only does metadata lookups against unique member names in Microsoft Analysis Services data sources, the match attribute allows you to specify the pattern such as "[Time].[Fiscal].[All Time Periods].[FY{0000}]". The rest of the patterns just add on to this as each higher generation member name incorporates the name of the lower generation.

DTD Elements and Attributes

This section describes the elements and their attributes in the TimeSchema XML DTD.

<timeSchemas>

This is the outmost element. It has no attribute. Inside <timeSchemas> you can have multiple timeSchema elements, one for each data source.

<timeSchema>

The time schema for each data source needed in the application should be defined inside the timeSchema element. It has the following attributes:

Attribute	Required?	Description
cube	Yes for MSAS and SAP BW	The name of the cube. Required for Microsoft Analysis Services and SAP BW data sources.
dataSource	Yes	The name of the data source.
name	Yes	The name for this time schema. This is the name used to do the lookup by the timeSchema tag.

Attribute	Required?	Description
type	Yes	Four valid types: • Normal1D • Normal2D • Weekly1D • Weekly2D In a "normal" TimeSchema the year corresponds exactly with the standard (Gregorian) calendar year: it will have 365 days unless it is a leap year. In a weekly TimeSchema, the year corresponds to 52 weeks except where otherwise noted. This weekly calendar is a common fiscal planning calendar. 1D indicates "one-dimensional," meaning all the members are found in one dimension of an MDB cube. A 2D calendar type is "two-dimensional," with the year members kept in one dimension and the rest of the members in another dimension. This split of dimensions is common practice in IBM DB2 OLAP Server or Hyperion Essbase cube implementation.
useAliases	No	true to use aliases. For IBM DB2 OLAP Server or Hyperion Essbase only. The default is false. When useAliases is set to true, make sure the pattern specified in the match attribute (in the <level> element) uses the aliases.

calculation

This element has the following attribute:

Attribute	Description
startDate	The start date for calculating the time table in the format of mm/dd/yyyy. For example: startDate="01/30/2000"

<exceptionYear>

Most commonly, when using a week-based time schema, the exceptionYear element is used to denote a 53-week year. Since each year is more than 52 weeks, it is necessary to mix in a 53-week year every 5 years or so. It can also be used to shorten a year if data is missing.

Attribute	Required?	Description
lengthDays	No	The number of Days in a year.
lengthWeeks	No	The number of Weeks in a year.

<dimension>

There can be at most two <dimension> elements in a TimeSchema. If the TimeSchema is one-dimensional (type of Normal1D or Weekly1D), there should only be one <dimension> element.

Attribute	Required?	Description
name	Yes	The name of the dimension where the members to use in the TimeSchema reside.
rootMember	No	The root member name.

<level>

The <level> element is nested within the <dimension> element for specifying how members in the specified generation in the dimension should be matched for each of the level type. It has the following attributes:

Attribute	Required?	Description
generation	Yes	The generation in the dimension that represents the specified type. See the type attribute of this element.
match	Yes	The pattern of the unique names to match. Specify the pattern in curly braces, with the following three special characters. • 0: a digit from 0 to 9. • #: an optional digit from 0 to 9. • M: an alphabetic character ([A-Z][a-z]). For example: `match="[Time].[Fiscal].[All Time Periods].[FY{0000}].[Qtr {0} FY{00}].{[MMM} FY {00}]"` will match members such as [Time].[Fiscal].[All Time Periods].[FY2002].[Qtr 1 FY02].[Jan FY02]. **Note:** When useAliases is set to true (in the <timeSchema> element), make sure the pattern specified in the match attribute (in the <level> element) uses the aliases.
order	No	Valid values are ascending or descending. The default is ascending, meaning that members ascend in time through the outline. In ascending order the year 1990 comes before 1991 and the month Jan comes before Feb. Use descending if the members are reversed in the outline for some reason.
startMember	No	The member the TimeSchema should start with. This member must match the pattern.
stopMember	No	The member the TimeSchema should stop at. This member must match the pattern.
type	Yes	Valid values are: • years • quarters • months • weeks

Chapter 22. Blox Portlet Tag Reference

The Blox Portlet Tag Library (`bloxPortlet.tld`) contains custom JSP tags that facilitate the creation of portlet links and action links. These tags allow you to attach a link to a Blox or Blox UI component that, when clicked, triggers an action or a portlet link. This chapter contains an overview of the tag library and reference material for tags in this library.

Blox Portlet Tag Library Overview

The `ClientLink` object of the Blox UI Model lets you attach a link to a specific Blox UI component. This URL-based link is handled by the browser when the component is clicked. However, in the portal environment, the link will only work the first time. After the link triggers a page reload, the portlet link becomes stale due to the way the portal server treats each page request. Subsequent clicking of the link will not submit a real action. The Blox Portlet Tag Library lets you add a PortletLinkDefinition or ActionLinkDefinition, which provides the following functionality:

- If a PortletLinkDefinition is added, it is then used to create a PortletLink object. The PortletLink object is used to define the actual link to invoke the URI with the specified parameter values. The URL is re-encoded by the portlet each time the page is refreshed, preventing it from getting stale.
- If an ActionLinkDefinition is added, it can be used to create a PortletLink. Or it can be used to obtain a portlet URI for this link by passing an action name to `BloxResponse.getActionURL()` .

When a PortletLinkDefinition or ActionLinkDefinition is combined with Blox, the definition is assigned to the Blox while the PortletLink or ActionLink is used to generate a ClientLink for use within the Blox UI Model. You can use the Blox Portlet tags inside any data presentation Blox, FormBlox, ReportBlox, or Blox UI components. While both PortletLinkDefinition and ActionLinkDefinition let you create a PortletLink, ActionLinkDefinition also lets you set the action name for this link definition to create the PortletURI. However, you cannot set the action name to pass to `BloxResponse.getActionURL()` after PortletLink is set. You will probably use action links in most cases as actions encapsulate the information used by the Portlet API's Property Broker to deliver property values provided by source portlets to target portlets.

Both PortletLinkDefinition and ActionLinkDefinition should be added to a top-level Blox. If you have a PresentBlox, the link should be added to the PresentBlox rather than the nested GridBlox.

Examples for Blox Portlet Tag Library

This section includes examples of the use of the Blox Portlet tags inside a GridBlox, a Button (a Blox UI component), a ReportBlox, and a TreeFormBlox. Each example demonstrates the basic approach to adding an action link or portlet link:

1. Add the `<bloxportlet:actionLinkDefinition>` tag inside the Blox or UI component to attach this link definition, and specify a name for the action using the `action` attribute.
2. Use the nested `<bloxportlet:parameter>` tag to specify the name of the parameter and its value.

You can then get to the PortletLink and set the link information or parameter values in a scriptlet. For details on the APIs, see the com.alphablox.blox.portlet package in the Javadoc documentation.

Adding a Link to a GridBlox

The following example defines an action named ″setTimeMember″ for a standalone GridBlox. Every time a notification event is sent when a UI Model component has been rebuilt, the action parameter ″memberName″ is set to the cell's value.

```
<%@ page contentType="text/html"%>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<%@ page contentType="text/html"
import="com.alphablox.blox.*,
        com.alphablox.blox.portlet.*,
        com.alphablox.blox.uimodel.*,
        com.alphablox.blox.uimodel.core.*,
        org.apache.jetspeed.portlet.*,
        java.lang.reflect.Method,
        com.alphablox.blox.uimodel.core.event.IEventHandler,
        com.alphablox.blox.uimodel.core.event.ComponentRebuiltNotify,
        com.alphablox.blox.uimodel.core.grid.GridCell,
        com.alphablox.blox.uimodel.core.grid.Grid"%>

<portletAPI:init/>

<%
    String gridBloxName = portletResponse.encodeNamespace("linkedGridBlox");
%>

<head>
  <blox:header />
</head>

<blox:grid id="gridBlox" bloxName="<%= gridBloxName %>" height="200"
   menubarVisible="false">
   <blox:toolbar visible="false" />
   <blox:data dataSourceName="Canned" />

   <bloxportlet:actionLinkDefinition action="setTimeMember">
      <bloxportlet:parameter name="memberName" />
   </bloxportlet:actionLinkDefinition>

<%
   BloxModel model = gridBlox.getBloxModel();
      Controller controller = model.getController();
      GridEventHandler eventHandler =
         new GridEventHandler(gridBlox.getPortletLink("setTimeMember"));

      controller.addEventHandler(eventHandler);
  %>
</blox:container>

<%!
public class GridEventHandler implements IEventHandler {
    private PortletLink portletLink;

    public GridEventHandler(PortletLink portalLink) {
        this.portletLink = portalLink;
    }

 public boolean handleComponentRebuiltNotify(ComponentRebuiltNotify event)
throws Exception {
        Component component = event.getComponent();
```

```
        if (component instanceof GridCell) {
            GridCell cell = (GridCell)component;
            if (cell.isRowHeader() && !cell.isColumnHeader()) {
                String cellValue = cell.getValue();

                portletLink.setParameterValue("memberName", cellValue);

                Component text = cell.get(0);
                text.setClientLink(portletLink.getClientLink());
            }
        }
  return false;
   }
    }
%
```

Additional code is needed to use this information. The following code sample shows an `actionPerformed()` method that is added to the Portlet Controller class for processing and utilizing this parameter value:

```
<%
  public void actionPerformed(ActionEvent event) throws PortletException {
     String actionString = event.getActionString();
     PortletRequest request = event.getRequest();

        if (actionString.equals("setTimeMember")) {
            String timeMember = request.getParameter("memberName");
            // ... use the time member accordingly ...
        }
  }
%>
```

Adding a Link to a Button

The following example defines an action named ″showData″ for a Button with three parameters. The PortletLink's ClientLink is hooked up with the button, so when the button is clicked, values for two of the three parameters for this PortletLink are set. Additional code is needed to use this information, such as in another portlet. This example only demonstrates how to set the link.

```
<%@ page contentType="text/html"%>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<%
  String bloxName = portletResponse.encodeNamespace("buttonContainer");
%>

<head>
  <blox:header />
</head>

<blox:container id="myButtonContainer" bloxName="<%= bloxName %>"
   width="40" height="20">
   <bloxportlet:actionLinkDefinition action="showData">
      <bloxportlet:parameter name="a" />
      <bloxportlet:parameter name="b" value="2" />
      <bloxportlet:parameter name="c" />
   </bloxportlet:actionLinkDefinition>

<%
   BloxModel model = myButtonContainer.getBloxModel();
   model.clear();
```

```
   Button myButton = new Button("button1", "Show Data");
   model.add(myButton);
   model.changed();

   // programmatically set the parameter values for the named Portlet
   PortletLink plink = myButtonContainer.getPortletLink("showData");
   plink.setParameterValue("a","1");
   plink.setParameterValue("c","xyz");
   myButton.setClientLink(plink.getClientLink());
%>
</blox:container>
```

Adding a Link to a ReportBlox

The following example defines an action link named selectProductCode for a ReportBlox. The link is attached to the Product column. When the product name in the report is clicked, the link sets the value for the parameter ″code″ to the product's code. Additional code is needed to use this information, such as in another portlet. This example only demonstrates how to set the link.

```
<%@ page contentType="text/html" %>
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxreporttld" prefix="bloxreport"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<head>
   <blox:header/>
   <link rel="stylesheet" href="/AlphabloxServer/theme/report.css">
</head>
<%
   String reportName = portletResponse.encodeNamespace("myReportBlox");
%>

<bloxreport:report id="report" bloxName="<%= reportName %>" interactive="false">
   <bloxreport:cannedData />
   <bloxreport:filter expression="Sales < 100"  />
   <bloxreport:group members="Area" />
   <bloxreport:sort member="Week_Ending"  />

   <bloxportlet:actionLinkDefinition action="selectProductCode">
      <bloxportlet:parameter name="code" />
   </bloxportlet:actionLinkDefinition>

<%
    PortletLink link = report.getPortletLink("selectProductCode");
    link.setParameterValue("code", "<value member=\"code\"/>");
    String href = link.getLinkHRef();

    String productLink = "<a href=\""+ href + "\"><value/></a>";
  %>
  <bloxreport:text>
    <bloxreport:data columnName="Product" text="<%= productLink %>" />
  </bloxreport:text>
</bloxreport:report>
```

Adding a Link to a TreeFormBlox

The following example defines two action links named selectItem and selectFolder for a TreeFormBlox. The PortletLink's ClientLink is hooked up with the TreeFormBlox, so when an item or a folder is clicked, the parameter value is set to the item name or folder name for this PortletLink. Additional code is needed to use this information, such as in another portlet. This example only demonstrates how to set the link.

```
<%@ page contentType="text/html"%>

<%@ taglib uri="bloxformtld" prefix="bloxform"%>
<%@ taglib uri="bloxportlettld" prefix="bloxportlet" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>

<portletAPI:init/>

<%
  String treeName = portletResponse.encodeNamespace("tree2");
%>

<head>
  <blox:header />
</head>
<%!
  private String getFolder(PortletLink folderLink, String label) {
    folderLink.setParameterValue("folder", label);
    return folderLink.getLinkHRef();
  }

  private String getItem(PortletLink itemLink, String label) {
    itemLink.setParameterValue("item", label);
    return itemLink.getLinkHRef();
  }
%>

<bloxform:tree id="tree" bloxName="<%= treeName %>"
   draggingEnabled="true">
   <bloxportlet:actionLinkDefinition action="selectItem">
      <bloxportlet:parameter name="item" />
   </bloxportlet:actionLinkDefinition>

   <bloxportlet:actionLinkDefinition action="selectFolder">
      <bloxportlet:parameter name="folder" />
   </bloxportlet:actionLinkDefinition>

   <%
    PortletLink itemLink = tree.getPortletLink("selectItem");
    PortletLink folderLink = tree.getPortletLink("selectFolder");
   %>

   <bloxform:folder name="root" label="Root Folder"
      href="<%= getFolder(folderLink, \"Root Folder\") %>">
      <bloxform:item label="Item1" href="<%= getItem(itemLink, \"Item1\") %>" />

      <bloxform:folder label="Folder1"
         href="<%= getFolder(folderLink, \"Folder1\") %>" >
         <bloxform:item label="Item2" href="<%= getItem(itemLink, \"Item2\") %>" />
         <bloxform:item label="Item3" href="<%= getItem(itemLink, \"Item3\") %>" />
      </bloxform:folder>

      <bloxform:folder label="Folder2"
         href="<%= getFolder(folderLink, \"Folder2\") %>" >
         <bloxform:item label="Item4" href="<%= getItem(itemLink, \"Item4\") %>" />
         <bloxform:item label="Item5" href="<%= getItem(itemLink, \"Item5\") %>" />
      </bloxform:folder>

      <bloxform:folder name="Folder3" label="Grow Tree"
         href="<%= getFolder(folderLink, \"Folder3\") %>" />

      <%
        tree.addFormEventListener(new TreeEventListener());
       %>
  </bloxform:treer>
```

The <bloxportlet:actionLinkDefinition> Tag

This tag adds an ActionLinkDefinition. It should be nested inside a data presentation Blox, a Blox UI component, a FormBlox, or a ReportBlox. It can nest one or more `<bloxportlet:parameter>` tag to pass one or more parameter names. It has the following attribute:

Attribute	Required	Description
`action`	Yes	The name of the action link.

The <bloxportlet:actionLink> Tag

This tag lets you specify the name of a previously defined action link. It can then nest one or more `<bloxportlet:parameter>` tag to pass one or more parameter values for the named action. It has the following attribute:

Attribute	Required	Description
`action`	Yes	The name of an action. This action name should have been defined in a `<bloxportlet:actionLinkDefinition>` tag.

The <bloxportlet:parameter> Tag

This tag lets you specify the value for the named parameter. It has the following tag attributes:

Attribute	Required	Description
`name`	Yes	The name of a parameter whose value is to be set either through the value tag attribute, or in Java code through the `PortletLink.setParameterValue(String` *`name`*`, String` *`value`*`)` method.
`value`	No	The value for the named parameter.

The <bloxportlet:portletLinkDefinition> Tag

This tag adds an PortletLinkDefinition. It should be nested inside a data presentation Blox, a Blox UI component, a FormBlox, or a ReportBlox. It can nest one or more `<bloxportlet:parameter>` tag to pass one or more parameter names. It has the following attribute:

Attribute	Required	Description
`name`	Yes	The name of the PortletLink.

The <bloxportlet:portletLink> Tag

This tag lets you specify a previously defined PortletLink. It can then nest one or more `<bloxportlet:parameter>` tag to specify the value of a named parameter in the named PortletLink. It has the following tag attribute:

Attribute	Required	Description
`name`	Yes	The name of a PortletLink previously defined using the `<bloxportlet:portletLinkDefinition>` tag.

Chapter 23. Blox UI Tag Reference

This chapter contains reference material for the Blox UI modifier tags in the `bloxui.tld` tag library. These tags allow you to perform powerful Blox user interface and data layout modification and customization in the DHTML client.

Blox UI Tags Overview

The DB2 Alphablox Tag Libraries provide custom tags to use in a JSP page for creating each Blox. It also includes a Blox UI Tag Library for modifying Blox UI and for adding custom analysis functionality all through the use of tags. These tags work directly with the DHTML user interface model and do not affect other clients.

With the `blox.tld` tag library, you can create and add a Blox to your page. With the `bloxui.tld` tag library, you can customize Blox appearance and behavior above and beyond the Blox properties you can set through the Blox tags. The Blox UI tags usually nest inside a presentation Blox tag as they customize those Blox appearances and behaviors.

Whenever possible, you should use Blox tags to set data properties, general user interface organization such as `chartFirst`, `menubarVisible`, and `splitPaneOrientation`, and general Blox features such as cell alerts and writeback that are available in all clients. Use the Blox UI tags only if you are using the DHTML client and if you need higher level of UI customization than is provided by Blox properties. These tags use styles that override the theme-based Cascading Stylesheet classes settings used in the DHTML client.

There are several types of Blox UI tags:

- Component Customization tags: These are tags for UI component customization, such as customizing menus and toolbars. All the common component names used by the Blox UI model are constants. You can find all the constants in the `ModelConstants` interface under the com.alphablox.blox.uimodel package in the Javadoc documentation. Using the component customization tags, you can identify the components by their names and then specify the values of their attributes such as their position, visibility, or style.
- Custom Layout tags: These are tags that allow primarily customization of grid layout, such as applying a butterfly layout or adding spaces among data columns or rows.

- Analysis tags: These are tags that add data analysis features to in your application.
- Utility tags: These are convenience tags to facilitate processing of actions.
- : This tag can be used to enhance the user experience for users with disabilities.

To use the Blox UI modifier tags, you need to include the following taglib import statement in your page:

```
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
```

Blox UI Tag Library Cross References

The Blox UI Tag Library contains the following tags:

Component Customization Tags

- "CalculationEditor Tag" on page 368
- "Component Tag" on page 369
- "Custom Menu Tags" on page 393
- "Custom Toolbar Tags" on page 400

Custom Analysis Tags

- "The <bloxui:bottomN> Tag" on page 374
- "The <bloxui:customAnalysis> Tag" on page 376
- "The <bloxui:topN> Tag" on page 380

Custom Layout Tags

- "The <bloxui:butterflyLayout> Tag" on page 382
- "The <bloxui:compressLayout> Tag" on page 384
- "The <bloxui:customLayout> Tag" on page 386
- "The <bloxui:gridHighlight> Tag" on page 386
- "The <bloxui:gridSpacer> Tag" on page 388
- "The <bloxui:title> Tag" on page 391 (also applies to PresentBlox and ChartBlox)

Utility Tags

- "The <bloxui:actionFilter> Tag" on page 407
- "The <bloxui:gridFilter> Tag" on page 409
- "The <bloxui:clientLink> Tag" on page 412
- "The <bloxui:setProperty> Tag" on page 412

"Accessibility Tag" on page 406
- "The <bloxui:accessibility> Tag" on page 406

These tags and their attributes are described in the following sections.

CalculationEditor Tag

This tag adds the Calculation Editor option to the right-click menu, the Data menu in the menu bar, and the Calculation Editor icon to the toolbar.

The Calculation Editor is a user interface that allows users to add new members by specifying the members involved in the calculation and the calculation expression. Various arithmetic and special calculation functions are available and users can

specify where the calculated member should be positioned, what generation level it should be, and how missing values should be treated in the calculation. When users select the Calculation Editor option, the Calculation Editor pops up. To see a sample of the Calculation Editor, bring up the Query Builder from the Assembly tab in the DB2 Alphablox home page. The Query Builder has a Calculation Editor icon added to the toolbar.

The `<bloxui:calculationEditor>` tag should be nested within a presentation Blox tag as follows:

```
<blox:present id="myPresentUI">
   <blox:data bloxRef="myDataBlox" />
   <bloxui:calculationEditor />
</blox:present>
```

This tag has no attributes.

Component Tag

Component is the base class for all UI model visual components. This class provides default behaviors and properties which are common across all visual components. You can find all the constants representing the components in the `ModelConstants` interface under the `com.alphablox.blox.uimodel` package in the Javadoc documentation. Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with the first letter of second and each subsequent word in uppercase. For your convenience, a list of all constants are available at "Model Constants and Their Values" on page 413.

The `<bloxui:component>` tag should be nested within a presentation Blox tag to modify the UI component of that Blox.

The complete `<bloxui:component>` tag is as follows:

```
<bloxui:component
   alignment=""
   bloxRef=""
   clickable=""
   disabled=""
   height=""
   name=""
   positionBefore=""
   style=""
   themeClass=""
   title=""
   tooltip=""
   valignment=""
   visible=""
   width="">

      <bloxui:clientLink
         features=""
         link=""
         target="" />

</bloxui:component>
```

The component tag can be used to customize a named `Menu`, `MenuItem`, `Toolbar`, and `ToolbarButton` since it is the base for all UI model visual components.

The <bloxui:component> Tag Attributes

Attribute	Required	Description
alignment	No	The horizontal alignment setting of the component. Valid values are left, center, and right.
bloxRef	No	References an existing Blox to apply the tag to when the tag is used outside of the Blox's tag. This allows dynamically setting a Blox's UI components.
clickable	No	Set this to false to disable interaction. The component is displayed but is not clickable. The default is true. This attribute needs to be set on the outmost user interface Blox. All nested user interface Blox will inherit the attribute. You cannot set the clickable attribute on a nested user interface Blox.
disabled	No	Set this to true to disable the named component. The component becomes greyed out. The default is false.
height	No	The height of this component in pixels.
name	Yes	The name of the component. To customize the default component in the Blox user interface, specify the value for the UI component. This component can be a Menu, a MenuItem, a Toolbar, a ToolbarButton, a Button, or any component that extends from the Component class. These constants representing the UI component are all in the ModelConstants interface in the com.alphablox.blox.uimodel package. The following examples show two ways to identify the View menu in the menu bar. `name="<%= ModelConstants.VIEW_MENU %>"` `name="viewMenu"`
style	No	The style to attach to the component. Overrides the default style or theme-based style for the component. For example, the following style sets the background color of the named component to black with no border. `style="background-color: black; border-style:none;"`
themeClass	No	Name of the Cascading Stylesheet classes.
title	Yes for custom menus	The displayed title for the component. Custom components added must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse-over.
valignment	No	The vertical alignment setting of the component. Valid values are top, center, and bottom. The default is center.
visible	No	The visibility of the component. When set to false, the component is not displayed. The default is true.
width	No	The width of this component in pixels.

Nested <bloxui:clientLink> Tag

This is a nested tag for multiple Blox UI tags. See "The <bloxui:clientLink> Tag" on page 412 for details.

Component Tag Examples

Example 1: Customizing a menu item

The following example demonstrates how existing MenuItems (helpHelp, helpAbout, toolsGridOptions, and chartMenu) are customized using the <bloxui:component> tag.

- The <bloxui:component> tag is nested inside a PresentBlox.
- The Help... menu item under the Help menu (name = "helpHelp") is removed by setting its visibility to false (visible="false").
- The About Alphablox menu item (name = "helpAbout") is modified to say "About This App..." (title="About This App...").
- The Grid Options... menu item (name = "toolsGridOptions") is disabled from the Tools menu (disabled="true").
- The Chart menu (name = "chartMenu") is repositioned to before the Data menu (positionBefore="dataMenu").

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<blox:data id="dataBlox" dataSourceName="TBC" useAliases="true"
   query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year, Scenario) Qtr1
Qtr2 <CHILD Scenario Sales !"/>
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="myPresentBlox" width="700" height="500" >
   <blox:data bloxRef="dataBlox" />
   <bloxui:component name="helpHelp" visible="false" />
   <bloxui:component name="helpAbout" title="About This App..."
      tooltip="About this application" />
   <bloxui:component name="toolsGridOptions" disabled="true" />
   <bloxui:component name="chartMenu" positionBefore="dataMenu" />
</blox:present>
</body>
</html>
```

This customization task can also be done using the <bloxui:menu> and <bloxui:menuItem> tags. See "Custom Menu Tags" on page 393 for more information.

Example 2: Dynamically setting visibility of UI components using the bloxRef attribute

The following example demonstrates how to let a user turn on or off the Standard toolbar interactively.

- Two HTML buttons are created—Hide Toolbar and Show Toolbar.
- Upon initial load, the choice parameter is null.
- When the user clicks one of the buttons, set the value for the action parameter, and reload the file (UITagBloxRef.jsp) with the appropriate action appended to the URL.
- Use the <bloxui:component> tag to set the visibility of the Standard toolbar.

```
<%--UITagBloxRef.jsp --%>
<%@ taglib uri='bloxtld' prefix='blox'%>
<%@ taglib uri='bloxuitld' prefix='bloxui'%>

<%--Check the choice parameter. Upon initial load, the choice
    is null.
--%>

<%
   String choice = request.getParameter( "choice" );
```

```
   if ( choice != null ) {
      if ( "showToolbar".equals( choice ) ) {
%>
         <bloxui:component bloxRef="tagBloxRefBlox"
            name="standardToolbar"
            visible="true" />
<%
      }
      else if ( "hideToolbar".equals( choice ) ) {
%>
         <bloxui:component bloxRef="tagBloxRefBlox"
            name="standardToolbar"
            visible="false" />
<%
      }
   return;
   }
%>

<blox:data id="dataBlox" dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW (\"All Locations\", \"Measures\") \"Central\" \"East\"
      \"West\" \"All Locations\" \"Gross Margin\" <CHILD \"Ratios\"
      <ASYM <COLUMN (\"Scenario\", \"All Time Periods\") \"Actual\"
      \"Actual\" \"Forecast\" \"Forecast\" \"2000.Q3\" \"2000.Q4\"
      \"2001.Q1\" \"2001.Q2\" !" />

<html>
<head>
<blox:header />
</head>

<body>
<!--Add two buttons to allow users to hide/show the toolbar
    When the button is clicked, reload the page with the
    choice parameter specified.
-->
<input type=button value="Hide Toolbar"
onclick="window.location.href='UITagBloxRef.jsp?render=dhtml&choice=hideToo
lbar'">

<input type=button value="Show Toolbar"
onclick="window.location.href='UITagBloxRef.jsp?render=dhtml&choice=showToo
lbar'">

<hr>
<blox:present id="tagBloxRefBlox" width="700" height="500" visible="true">
   <blox:data bloxRef="dataBlox" />
</blox:present>
</body>
</html>
```

Example 3: Setting a PresentBlox Unclickable

The following example demonstrates how to make a user interface Blox non interactive using the `<bloxui:component` tag's `clickable` attribute.

- The `<bloxui:component>` tag is nested inside a PresentBlox with the component's `name` pointing to the name of the PresentBlox.
- You can only set the `clickable` attribute on the outmost UI Blox. In this example, the `clickable` attribute is set on the PresentBlox. You cannot set the `clickable` attribute on the nested GridBlox or ChartBlox inside the PresentBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
<blox:data id="dataBlox"
```

```
   dataSourceName="QCC-Essbase" useAliases="true"
   query="<SYM <ROW (\"All Products\") <CHILD \"All Products\"
      <COL (\"All Time Periods\") <CHILD \"All Time Periods\"
      <PAGE(Measures) Sales !" />
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="notclickablePresentBlox"
   width="80%" height="70%" menubarVisible="false">
   <blox:toolbar visible="false" />
   <blox:data bloxRef="dataBlox" />
   <bloxui:component name="notclickablePresentBlox" clickable="false" />
</blox:present>
</body>
</html>
```

Custom Analysis Tags

Custom analysis tags allow you to add custom analytical functionality to your grids and charts. These tags include:

- The <bloxui:bottomN> Tag
- The <bloxui:customAnalysis> Tag
- The <bloxui:eightyTwenty> Tag
- The <bloxui:percentOfTotal> Tag
- The <bloxui:topN> Tag

Once you add these tags in your presentation Blox tags (PresentBlox, GridBlox, or ChartBlox), these custom analytical functions show up in the right-click menu and the menu bar's Data menu under the Advanced option. For example, the following tags add six advanced data analysis options:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present ...>
   <bloxui:topN number="10" showRank="true" />
   <bloxui:topN number="5" showRank="true" />
   <bloxui:topN prompt="true" showRank="true"  number="20"/>
   <bloxui:bottomN number="10" showRank="true" />
   <bloxui:bottomN number="5" showRank="true" />
   <bloxui:bottomN prompt="true"  showRank="true"  number="7"/>
</blox:present>
</body>
</html>
```

The six added options show up in the right-click menu and the menu bar's Data menu.

Tip: Only one analysis operation can be in effect at a time. If the user chooses Top 10 first and then Bottom 5 or Percent of Total, each is an independent operation and does not retain the result from the previous operations.

The <bloxui:bottomN> Tag

The following shows all tag attributes for the `<bloxui:bottomN>` tag. This tag should be nested within the tag of a PresentBlox or GridBlox:

```
<bloxui:bottomN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

where:

Attribute	Required	Default	Description
`description`	No	Bottom *N;* Bottom N...	Specifies the text for this menu item under the Advanced menu. The default is "Bottom *N*" where *N* is the value of the `number` attribute. If the `prompt` attribute is set to `true`, the default is "Bottom N...."
`hideOthers`	No	`all`	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: `all`: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. `none`: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. `unranked`: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.
`membersToAnalyze`	No	`leaf`	Specifies whether to rank only the currently displayed members or all the leaf members. Valid values are: • `displayed`: ranks only the currently displayed members. • `leaf`: ranks all leaf members.
`name`	No		The menu ID for this analysis under the Data > Advanced menu option in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.

Attribute	Required	Default	Description
number	No	10	Specifies the number of members with the lowest values to show. If a number is not specified, the option "Bottom 10" will be displayed in the menu. When the prompt attribute is set to true, value for this attribute will become the default number shown in the pop-up dialog, prompting for the number of members to show.
preserveGrouping	No	true	Specifies whether to preserve grouping when ranking. When true, ranking is calculated per group when there are multiple dimensions on the axis. See "Example 1: Bottom 10 Analysis" on page 375.
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to enter the number of members of the lowest values they want to see. To specify the default value for the prompt, use the number attribute.
showOtherSummary	No	false	Specifies whether to add an "Other" row/column as a summary for the remaining, un-ranked members.
showRank	No	true	Specifies whether to show the ranking in an added column/row. If false, the members are sorted according to the rank without the added column/row showing the ranking

bottomN Tag Examples

Example 1: Bottom 10 Analysis

The following code will add a Bottom 10 advanced analysis option to the right-click menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:bottomN
     number="10"
     showRank="true" />
</blox:grid>
```

When a user selects this option, a column (or row, depending on whether the user selects a row header or column header) called "[member name] Bottom 10" will be added to the grid.

Example 2: Bottom N Analysis with Prompt

By default, unranked members on both the column and row axes are hidden unless hideOthers is set to none or unranked. To allow users to specify ranking

options such as the number of members they want to rank and whether to rank only the currently displayed members or leaf members only, set the `prompt` attribute to `true`:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
<blox:grid ...>
   <bloxui:bottomN
     prompt="true"
     number="7"
     showRank="true" />
</blox:grid>
```

When users choose this option, a dialog pops up with the value set in `number` being the default number of members to display.

Example 3: Bottom 5 And Other

The following code will add a Bottom 5 and Other menu option to the right-click menu:

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:bottomN
     description="Bottom 5 and Other"
     number="5"
     hideOthers="all"
     showOtherSummary="true"
     showRank="true" />
</blox:grid>
```

When a user selects this option, a column (or row, depending on whether the user selects a row header or column header) called "[member name] Bottom 5" will be added to the grid, with an added "Other" member on the opposite axis that provides summary value for the remaining non-ranked members. Note that `hideOthers` is set to `all` (the default) so the unranked members and members not involved in the calculations from both the column and row axes are not displayed.

If `hideOthers` is set to `unranked`, then only unranked members are hidden. Members on the opposite axis that are not involved in the calculation remain in the grid:

The <bloxui:customAnalysis> Tag

The following shows all tag attributes for the `<bloxui:customAnalysis>` tag. This tag should be nested within the tag of a PresentBlox or GridBlox:

```
<bloxui:customAnalysis
   analysis=""
   name="" />
```

where:

Attribute	Required	Description
analysis	Yes	Specifies the custom analysis object. For example: `<bloxui:customAnalysis` `analysis="<%= new TopN() %>" />` The custom analysis object (TopN in the above example) should implement AbstractAnalysis in the com.alphablox.blox.uimodel.tags.analysis package.
name	No	The menu ID for this analysis under the Data > Advanced menu option in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.

The <bloxui:eightyTwenty> Tag

The following shows all tag attributes for the <bloxui:eightyTwenty> tag. This tag should be nested within the tag of a PresentBlox and GridBlox.

```
<bloxui:eightyTwenty
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt="" />
```

where:

Attribute	Required	Default	Description
description	No	80/20 Analysis	Specifies the text for this menu item under the Advanced menu. If the prompt attribute is set to true, the default is "80/20 Analysis"
hideOthers	No	all	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: all: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. none: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. unranked: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.

Attribute	Required	Default	Description
membersToAnalyze	No	leaf	Specifies whether to include only the currently displayed members or all the leaf members in the calculation. Valid values are: • displayed: includes only the currently displayed members. • leaf: includes all leaf members.
name	No		The menu ID for this analysis under the Data > Advanced menu option in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.
number	No	80	Specifies the top percentage of data to show. When the prompt attribute is set to true, the default value 80 is shown in the pop-up dialog unless the number attribute is set differently.
preserveGrouping	No	true	Specifies whether to perform the calculation per group or regardless of groups. When true, calculation is performed per group when there are multiple dimensions on the axis.
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to set the options.

The <bloxui:percentOfTotal> Tag

The <bloxui:percentOfTotal> tag calculates the total of all the members and the percentage of each member and display them. It needs to be added within the tag of a PresentBlox or GridBlox. It has the following attributes:

```
<bloxui:percentOfTotal
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

where:

Attribute	Required	Default	Description
description	No	Percent of Total	Specifies the text for this menu item under the Advanced menu. If the prompt attribute is set to true, the default is "Percent of Total..."

Attribute	Required	Default	Description
hideOthers	No	all	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: all: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. none: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. unranked: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.
membersToAnalyze	No	leaf	Specifies whether to include only the currently displayed members or all the leaf members in the calculation. Valid values are: • displayed: includes only the currently displayed members. • leaf: includes all leaf members.
name	No		The menu ID for this analysis under the Data > Advanced menu option in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.
number	No	100	Specifies the percentage of data to show. When the prompt attribute is set to true, the default value 100 is shown in the pop-up dialog unless the number attribute is set differently.
preserveGrouping	No	true	Specifies whether to perform the calculation per group or regardless of groups. When true, calculation is performed per group when there are multiple dimensions on the axis.
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to set the options.

percentOfTotal Tag Example

The following example adds a "Percent of Total" option to the right-click menu and the menu bar's Data menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:percentOfTotal/>
</blox:grid>
```

The <bloxui:topN> Tag

This <bloxui:topN> tag should be nested within the tag of a PresentBlox or GridBlox. It has the following tag attributes:

```
<bloxui:topN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
   showRank=""
   showOtherSummary=""
/>
```

where:

Attribute	Required	Default	Description
description	No	Top *N;* Top N...	Specifies the text for this menu item under the Advanced menu. The default is "Top *N*" where *N* is the value of the number attribute. If the prompt attribute is set to true, the default is "Top N...."
hideOthers	No	all	Specifies whether the remaining non-ranked members and remaining members not involved in the calculation should be hidden from view. Valid values are: all: the default; hiding non-ranked members and members not involved in the calculations from both the column and the row axes. none: keep all the non-ranked members and members not involved in the calculations from both the column and row axes. unranked: hide only the non-ranked members; keep the members not involved on the opposite axis in the calculations.
membersToAnalyze	No	leaf	Specifies whether to rank only the currently displayed members or all the leaf members. Valid values are: • displayed: ranks only the currently displayed members. • leaf: ranks all leaf members.
name	No		The menu ID for this analysis under the Data > Advanced menu option in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.

Attribute	Required	Default	Description
number	No	10	Specifies the number of members with the highest values to show. If a number is not specified, the option "Top 10" will be displayed in the menu. When the prompt attribute is set to true, value for this attribute will become the default number shown in the pop-up dialog, prompting for the number of members to show.
preserveGrouping	No	true	Specifies whether to rank the members per group or regardless of the group. When true, calculation is performed per group when there are multiple dimensions on the axis.
prompt	No	false	Specifies whether to prompt for a number. When prompt is set to true, a dialog pops up, prompting the user to enter the number of members of the highest values they want to see. To specify the default value for the prompt, use the number attribute.
showOtherSummary	No	false	Specifies whether to add an "Other" row/column as a summary for the remaining, unranked members.
showRank	No	true	Specifies whether to show the ranking in an added column/row. If false, the members are sorted according to the rank without the added column/row showing the ranking.

topN Tag Example

The following code will add a Top 10 and Other advanced analysis option to the right-click menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:grid ...>
   <bloxui:topN
     description="Top 10 and Other"
     number="10"
     hideOthers="all"
     showOtherSummary="true"
     showRank="true" />
</blox:grid>
```

When a user selects this option, a column (or row, depending on whether the user selects a row header or column header) called "[member name] Top 10" will be added to the grid, with an added "Other" member on the opposite axis that provides summary value for the remaining non-ranked members. Note that hideOthers is set to all (the default) so:

- The unranked members are not displayed in addition to the added "Other" member, which may result in confusion.

- Members not involved in the calculation (such as Budget and Variance in the Scenario dimension) are hidden.

If you only want to hide the unranked member and keeping Budget and Variance in the Scenario dimension in the above example, then set `hideOthers` to `unranked`.

Custom Layout Tags

The custom layout tags work primarily on the GridBlox user interface in the DHTML client. These tags allow you to customize the layout of your grid by placing row or column headers in the middle or center of the grid, highlighting certain rows or columns, adding blank rows or columns, and more. The layout tags include the following:

- "The <bloxui:butterflyLayout> Tag" on page 382
- "The <bloxui:compressLayout> Tag" on page 384
- "The <bloxui:customLayout> Tag" on page 386
- "The <bloxui:gridHighlight> Tag" on page 386
- "The <bloxui:gridSpacer> Tag" on page 388
- "The <bloxui:title> Tag" on page 391 (also applies to PresentBlox and ChartBlox)

Except for the `<bloxui:title>` tag, these tags should be nested within a `<blox:present>` or a standalone `<blox:grid>` tag.

The <bloxui:butterflyLayout> Tag

This tag lets you position the row header column in the specified location in the grid. It has the following tag attributes. This tag should be nested within the tag of a `<blox:present>` or `<blox:grid>` tag.

```
<bloxui:butterflyLayout
   addSeparatorColumns=""
   applyLayout=""
   description=""
   name=""
   position=""
   scope=""
   separatorWidth=""
   showOnLayoutMenu="" />
```

where:

Attribute	Required	Description
`addSeparatorColumns`	No	When set to `true`, this adds an empty column between the header column and the data columns on its two sides. The default is `false`.
`applyLayout`	No	`true` — apply this layout when the page is loaded; `false` — do not apply this layout when the page is loaded. The default is `true`. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the `showOnLayoutMenu` attribute to `true` and turn on menu bar in your presentation Blox.
`description`	No	The display layout name when `showOnLayoutMenu` is set to `true`. The default value is "`Butterfly`."

Attribute	Required	Description
name	No	The menu ID for this layout under the Format menu in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.
position	No	Specifies whether the header column should be added before or after the scope specified. Valid values are before and after. The default is before.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menu bar (if it does not exist already), with the Butterfly menu option under the Layout submenu. The default is false.
scope	Yes	Defines the members relative to which the header column should be displayed. The following example has the header column positioned before Forecast: `scope="Forecast"` `postion="before"` To place the header column before a tuple, separate scope members with semicolons and enclose the entire scope in curly brackets as shown in the following example: `scope="{Forecast; Qtr 1 01}"` `position="before"` Specification of the tuple preserves the relative position of the header column to the tuple. In the example above where the tuple {Forecast; Qtr 1 01} is specified, when users drill down on Qtr 1 01, the tuples {Forecast; Jan 01}, {Forecast; Feb 01}, and {Forecast; Mar 01} will be displayed to the left of the row column header while {Forecast; Qtr 1 01} is displayed to the right of the header. For scoping syntax for various multidimensional and relational data sources, see the scope attribute discussion in “cellAlert” on page 223.
separatorWidth	No	Sets the width of the separator columns in pixels. The default is 10 pixels.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menu bar (if it does not exist already), with the Butterfly menu option under the Layout submenu. The default is false.

You may want to limit the data navigation functions allowed in a grid displayed in a butterfly layout since the layout may become irrelevant if the data is changed. For example:

- If users choose to hide the tuple specified as the scope, the layout cannot be applied. Instead of a butterfly layout, users will see a normal grid with row header columns on the left.
- The <bloxui:butterflyLayout> tag only supports column-based layout (vertical butterfly), with row headers in the middle and data to the left and right. The format may be lost if users choose to pivot or swap axes (no horizontal butterfly).

- The row header column is always displayed based on the relative position to the scope you specified. If you specify the header column to appear before the tuple {Forecast; Qtr 1 01}, when users drill down on Qtr 1 01, the tuples {Forecast; Jan 01}, {Forecast; Feb 01}, and {Forecast; Mar 01} will appear to the left of the row column header while {Forecast; Qtr 1 01} is displayed to the right of the header.

Since this tag works on the grid UI to display in a specific layout, it may be confusing to users if the layout disappears after some interaction with the grid. You can discourage data navigation by turning off toolbar and menu bar. You can also disable data navigation functions using the `<bloxui:menu>` or `<bloxui:component>` tag, or remove certain actions by setting the common Blox property removeAction. Or you can catch the events and display a message box informing the users that the actions are not supported. See the *Developer's Guide* for additional information and an example.

butterflyLayout Tag Example

The following example applies the butterfly layout to the grid with the header column positioned before the member Forecast. This also adds a Butterfly layout menu option to the menu bar's Format menu.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
...
<blox:present id="myPresent" width="600" height="500" >
   ...
   <bloxui:butterflyLayout scope="Forecast"
      showOnLayoutMenu="true"/>
...
</blox:present>
```

The <bloxui:compressLayout> Tag

This tag lets you compress the column and row headers into one level when there are multiple dimensions on the column or row axes. It has the following tag attributes. This tag should be nested within the tag of a `<blox:present>` or `<blox:grid>` tag.

```
<bloxui:compressLayout
   applyLayout=""
   compressColumns=""
   compressRows=""
   description=""
   memberSeparator=""
   name=""
   showOnLayoutMenu="" />
```

where:

Attribute	Required	Description
`applyLayout`	No	`true` — apply this layout when the page is loaded; `false` — do not apply this layout when the page is loaded. The default is `true`. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the `showOnLayoutMenu` attribute to `true` and turn on menu bar in your presentation Blox.

Attribute	Required	Description
compressColumns	No	true — to compress the column headers into one level. If you have multiple dimensions on the column axis, the headers can be compressed into one level. The default for compressColumns is false. When this attribute is set to true, the default members separator is a vertical bar (" | "). **Note:** If this or the compressRows attributes are not specified, the tag will not do anything.
compressRows	No	true — to compress the column headers into one level. If you have multiple dimensions on the row axis, the headers will be compressed into one level. The default for compressRows is false. When this attribute is set to true, the default members separator is a vertical bar (" | "). **Note:** If this or the compressColumns attributes are not specified, the tag will not do anything.
description	No	The display layout name when showOnLayoutMenu is set to true. The default value is "Compressed Headers."
memberSeparator	No	The text to use to separate the members. The default value is a vertical bar (" | ") with a space before and after.
name	No	The menu ID for this layout under the Format menu in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menu bar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the description attribute. The default is false.

compressLayout Tag Example

The following example compresses the row and column headers in the grid using " / " as the member separator. This also adds a Compressed Layout menu option to the menu bar's Format menu.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>
...
<body>
<blox:present id="myPresent" width="600" height="500" >
   ...
   <bloxui:compressLayout
       compressRows="true"
       compressColumns="true"
       showOnLayoutMenu="true"
       memberSeparator = " / " />
...
</blox:present>
...
</body>
</html>
```

Note: When column or row headers are compressed, all the model components are copied into a single grid header cell. For example, when Actual and Qtr 3 01 are compressed using ″ / ″ as the separator, the three Static components are placed into a single cell.

The <bloxui:customLayout> Tag

This tag lets you custom grid layouts. It has the following tag attributes. This tag should be nested within the tag of a <blox:present> or <blox:grid> tag.

```
<bloxui:customLayout
   applyLayout=""
   layout=""
   name=""
   showOnLayoutMenu=""/>
```

where:

Attribute	Required	Description
applyLayout	No	true — apply this layout when the page is loaded; false — do not apply this layout when the page is loaded. The default is true. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the showOnLayoutMenu attribute to true and turn on menu bar in your presentation Blox.
layout	Yes	Specifies the layout object. For example: `<bloxui:customLayout layout="<%= new ButterflyLayout() %>" />` The name of the class with the custom layout. Can be a custom class that implements the AbstractLayout class in the com.alphablox.blox.uimodel.tags.layout package, or an existing class in the package.
name	No	The menu ID for this layout under the Format menu in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menu bar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the layout attribute. The default is false.

The <bloxui:gridHighlight> Tag

This tag lets you highlight a member or members by specifying the scope and the style to use. It has the following tag attributes. This tag should be nested within the tag of a <blox:present> or <blox:grid> tag.

```
<bloxui:gridHighlight
   applyLayout=""
   description=""
   includeData=""
   includeHeaders=""
   name=""
   scope=""
   selection=""
   showOnLayoutMenu=""
   style="" />
```

where:

Attribute	Required	Description
applyLayout	No	true — apply this layout when the page is loaded; false — do not apply this layout when the page is loaded. The default is true. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the showOnLayoutMenu attribute to true and turn on menu bar in your presentation Blox.
description	No	The display layout name when showOnLayoutMenu is set to true.
includeData	No	true — include data cells in the specified scope; false — exclude data cells in the specified scope; the default is true.
includeHeaders	No	true — include header cells in the specified scope; false — exclude header cells in the specified scope; the default is true.
name	No	The menu ID for this grid highlight layout under the Format menu in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.
scope	Yes. Otherwise, specify the selection, or the tag will not do anything.	Defines the members to be highlighted. Separate the members in the scope using semicolons, enclosed with curly brackets. The following example has the gross margin for the West region to be highlighted: `scope="{West;Gross Margin}"` For scoping syntax for various multidimensional and relational data sources, see the scope attribute discussion in "cellAlert" on page 223.
selection	Yes. Otherwise, specify the scope, or the tag will not do anything.	Specifies either rowHeaders or columnHeaders to be highlighted. This allows you to customize the style for all row headers or column headers.
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menu bar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the description attribute. The default is false.
style	Yes. Otherwise the theme-based style is applied as usual and the tag will have no effect.	The style to attach to the grid highlight. Overrides the default style or theme-based style for the component. For example, the following style sets the background color of the highlighted cells to black with no border. `style="background-color: black; border-style:none;"`

gridHighlight Tag Example

The following example highlights the column headers in the grid with black text on yellow background color when the page is loaded. Another layout to highlight gross margin for the West region is not applied upon page load (`applyLayout="false"`). Users can choose to apply this layout from the Format menu in the menu bar.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>
...
<html>
<head>
   <blox:header />
</head>
<body>
...
<blox:present id="myPresent" width="600" height="500" >
   <blox:data dataSourceName="myData" query="<your query here>"... />
   <bloxui:gridHighlight
      description="Highlight Column Headers"
      selection="columnHeaders"
      style="color: black; background-color: yellow;"
      showOnLayoutMenu="true"/>
   <bloxui:gridHighlight
      description="Highlight West Gross Margin"
      scope="{west;gross margin}"
      style="font-weight:bold; color: teal; background-color: #FFFF99;"
      showOnLayoutMenu="true"
      applyLayout="false"/>
...
</blox:present>
</body>
</html>
```

The <bloxui:gridSpacer> Tag

This tag lets you add space between columns or rows. It has the following attributes. This tag should be nested within the tag of a `<blox:present>` or `<blox:grid>` tag.

```
<bloxui:gridSpacer
   applyLayout=""
   axis=""
   description=""
   height=""
   locked=""
   name=""
   position=""
   scope=""
   showOnLayoutMenu=""
   style=""
   width="" />
```

where:

Attribute	Required	Description
`applyLayout`	No	`true` — apply this layout when the page is loaded; `false` — do not apply this layout when the page is loaded. The default is `true`. Sometimes you may not want a layout to be applied until the user chooses to. In this case, set the `showOnLayoutMenu` attribute to `true` and turn on menu bar in your presentation Blox.
`axis`	Yes	Specifies `column` or `row` to add a spacer.
`description`	No	The display layout name when `showOnLayoutMenu` is set to `true`.
`height`	No	Number of pixels for the height of the spacer when a spacer is added to the row axis (`axis="row"`). The default is 10 pixels.

Attribute	Required	Description
locked	No	When set to true, this locks the spacer location on the screen so it does not scroll outside of the viewing area. Since the space added is actually an empty row/column, if locked is set to false, this empty row/column will scroll as normal data rows/columns. This setting only applies when the spacer is added with a position of before or after. The default is false.
name	No	The menu ID for this grid spacer layout under the Format menu in the menu bar. Assigning a name gives you programmatic access to the menu for custom event handlers and actions.
position	Yes	Specifies the position for the spacer. Valid values are: • after: adds a spacer after the named member. Use the scope attribute to specify the member. • before: adds a spacer before the named member. Use the scope attribute to specify the member. • bottom: adds a spacer to the bottom of the grid. Attribute axis should be set to row. • interlace: adds a spacer between columns or rows. • left: adds a spacer to the left of the grid. Attribute axis should be set to column. • memberchange: adds a spacer when member in the specified dimension (through the scope attribute) changes. • right: adds a spacer to the right of the named axis. Attribute axis should be set to column. • top: adds a spacer to the top of the grid. Attribute axis should be set to row. • A number: adds a spacer at the Nth column or row. For example, the following code adds a spacer as the third rows (0 being the top): position="2" axis="row" A value of 0 is the same as adding a spacer to the top or left of the grid. The axis attribute needs to be set for this to work. When position is set to memberchange, a scope has to be specified.
scope	Yes	Defines the scope containing the dimension to use when position is set to memberchange. The following example specifies to add a spacer whenever the member in Forecast is changed: scope="Forecast" position="memberchange" For scoping syntax for various multidimensional and relational data sources, see the scope attribute discussion in “cellAlert” on page 223.

Attribute	Required	Description
showOnLayoutMenu	No	When set to true, this adds a Format menu to the menu bar (if it does not exist already), with an added menu option under the Layout submenu, labeled after the value of the description attribute. The default is false.
style	No	The style to attach to the spacer. Overrides the default style or theme-based style for the component. For example, the following style sets the background color of the spacer to black with no border. `style="background-color: black; border-style:none;"` If no style is specified, the theme-based style is used.
width	No	Number of pixels for the width of the spacer when a spacer is added to the column axis (axis="column"). The default is 10 pixels.

gridSpacer Tag Example

The following example adds six spacers to the grid—top border, bottom border, left border, right border, column separator, and Location separator.

- The top and bottom borders are added with the axis attribute set to row and the position attribute set to top and bottom.
- The left and right borders are added with the axis attribute set to column and the position attribute set to left and right.
- The column separators are added with the axis attribute set to column and the position set to interlace, so there is a spacer between every two columns.
- The Location separator is added to create a grouping effect by adding a spacer when the member in the All Locations dimension changes. This is done by setting the axis to row, the position to memberchange, and the scope to All Locations.

```
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<blox:data id="gridSpacerData"
   dataSourceName="qcc-essbase" useAliases="true" visible="false"
   query="<ROW ('All Locations', 'Measures')  'Central'  'East'  'West'
'All Locations' 'Gross Margin' <CHILD 'Ratios' <ASYM <COLUMN ('Scenario',
'All Time Periods')     'Actual'  'Actual'  'Forecast'  'Forecast'
'2000.Q3'  '2000.Q4'  '2001.Q1'  '2001.Q2'!" />

<html>
<head>
   <blox:header />
</head>
<body>
<blox:grid id="gridSpacer" width="80%" height="500" visible="true">
   <blox:data bloxRef="gridSpacerData" />
   <bloxui:toolbar name="standardToolbar" visible="false" />

   <bloxui:gridSpacer
      axis="column"
      position="right"
      width="5"
      style="background-color: red;"
      description="Right Border"
      showOnLayoutMenu="true" />
```

```
   <bloxui:gridSpacer
      axis="column"
      position="left"
      width="5"
      style="background-color: red;"
      description="Left Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="top"
      height="5"
      style="background-color: red;"
      description="Top Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="bottom"
      height="5"
      style="background-color: red;"
      description="Bottom Border"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="column"
      position="interlace"
      width="2"
      style="background-color: yellow;"
      description="Column Separators"
      showOnLayoutMenu="true" />

   <bloxui:gridSpacer
      axis="row"
      position="memberchange" scope="All Locations"
      description="Location Separators"
      height="5"
      showOnLayoutMenu="true" />
</blox:grid>
</body>
</html>
```

The <bloxui:title> Tag

The `<bloxui:title>` tag allows you to add a title to the top of a presentation Blox (PresentBlox, GridBlox, and ChartBlox). The benefit of using this tag to add a title as opposed to using general HTML tags is better integration into the Blox user interface. Because the title becomes part of the presentation Blox, it automatically inherits the style applied to the Blox and wraps as the Blox is re-sized in the browser.

The `<bloxui:title>` tag should be added inside a presentation Blox. It contains the following tag attributes:

```
<bloxui:title
   title=""
   style=""
   alignment="" />
```

where:

Attribute	Description
`title`	The title to display.

style	The style to apply to the title. For example, style="font-family: Arial; font-weight: bold; font-size: 14pt; color: black; background-color: #FFFFCC;"
alignment	Alignment for the title. Valid values are left, center, and right.

The style defined for the title only applies to the title text rather than the entire rendered table cell. If you want to specify a background color for the title, you should also make sure that the background color of the presentation Blox use the same color.

To set the background color for presentation Blox, use the <bloxui:component> tag. This is demonstrated in the following example.

title Tag Example

The following example shows how to set the title of a GridBlox:

- The <bloxui:title> tag is nested inside a GridBlox.
- The background color, text color, font size and font style is set using the style attribute.
- The GridBlox's background color is set to the same background color as that of the title using the <bloxui:component> tag, with the component name set to the name of the GridBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<blox:data id="myDataTest"
   dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW ('All Locations', 'Measures') 'Central' 'East'
      'West' 'All Locations' 'Gross Margin' <CHILD 'Ratios'
       <ASYM <COLUMN ('Scenario', 'All Time Periods') 'Actual'
      'Actual'  'Forecast' 'Forecast' '2000.Q3' '2000.Q4' '2001.Q1'
      '2001.Q2'!" />

<html>
<head>
   <blox:header />
</head>

<blox:grid id="myGridTest"
   width="80%"
   height="80%"
   visible="true"
   menubarVisible="false"
   bandingEnabled="true"
   gridLinesVisible="false">
   <blox:data bloxRef="myDataTest" />
   <bloxui:component name="navigationToolbar" visible="false"/>
   <bloxui:component name="standardToolbar" visible="false"/>

   <bloxui:component name="myGridTest"
      style="background-color: #FFFFCC; border-style:none;" />

   <bloxui:title title="Sales and Gross Margin By Location - FY'02"
      style="font-family: Arial; font-weight: bold;
         font-size: 14pt; color: black; background-color: #FFFFCC;"
      alignment="center" />
```

```
</blox:grid>
</body>
</html>
```

Custom Menu Tags

The UI tags for customizing the menu bar allow you to add, remove, or disable menus and menu items to the default menu bar in a PresentBlox, GridBlox, or ChartBlox. Its tags need to be nested inside the tags for these user interface Blox. By default, the menu bar is available in a PresentBlox, a standalone GridBlox, or a standalone ChartBlox (`menubarVisible="true"`).

This section describes the general concepts related to the `Menubar`, `Menu`, and `MenuItem` components and provides tag reference for these components.

Menubar, Menu, and MenuItem

Each of the menus in the menu bar is a Menu component that you can remove, disable, or reposition. Each menu has menu items. Each menu item can also be removed, disabled, or repositioned. In addition, you can add your custom menus and menu items to the menu bar, or customize the operations associated with the menu items.

To specify actions associated with the menu items, you can use the `<bloxui:clientLink>` tag to load a URL or call a JavaScript function. You can also invoke server-side code through the `<bloxui:actionFilter>` tag. See "The <bloxui:actionFilter> Tag" on page 407 for more information.

Menu Tags Listing

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:menu
   name=""
   bloxRef=""
   accesskey=""
   disabled=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible=""
   >
   <bloxui:menuItem
      name=""
      accesskey=""
      checkable=""
      checked=""
      disabled=""
      imageURL=""
      positionBefore=""
```

```
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible=""
      >
      <bloxui:clientLink
         features=""
         link=""
         target="" />
   </bloxui:menuItem>
</bloxui:menu>
```

<bloxui:menu> Tag Attributes

Attribute	Required	Description
name	Yes	The name of the menu. If a Menu with the specified name is found, then the tag acts on the component. Otherwise, a new Menu is created. To customize the default menus in the menu bar, specify one of the following valid values: • bookmarkMenu • chartMenu • dataMenu • editMenu • fileMenu • helpMenu • toolsMenu • viewMenu Or you can specify the value using the constants. The following examples show two ways to specify the Tools menu in the menu bar. `name="<%= ModelConstants.TOOLS_MENU %>"` `name="toolsMenu"`
bloxRef	No	References an existing Blox to apply the tag to when the tag is used outside of the Blox's tag. See "Example on bloxRef Attribute" on page 371.
accesskey	No	The accelerator key. The specified character is used to search through the title of this menu. If it is found, the character is underlined in the menu's title in the user interface, indicating that it is the keyboard accelerator for the menu. **Note:** This only works on nested menus, not the top-level menus in the menu bar.
disabled	No	Set this to true to disable a menu. The menu is displayed in the menu bar but is greyed out.
positionBefore	No	The position before which the menu should be displayed. If this is not specified, newly added menu is appended to the end of the menu bar. To position the menu before the DB2 logo, set the value of value of this attribute to logo: `positionBefore="logo"`

Attribute	Required	Description
resourceName	No	Loads the named resource file into the component. This allows you to have a menu tag that creates a new menu from a menu XML file. See Chapter 25, "XML Resource Files Reference," on page 421 for more information.
title	Yes for custom menus	The displayed title for the menu. Custom menus added to the menu bar must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse-over.
visible	No	The visibility of the menu. When set to false, the menu is not displayed in the menu bar. The default is true.

Nested <bloxui:menuItem> Tag Attributes

Attribute	Required	Description
name	Yes	The name of the menu item. Specify your custom menu item names to add a custom menu item. If a MenuItem with the specified name is found, then the tag acts on the component. Otherwise, a new MenuItem is created. To customize a built-in menu item, see "Built-in Menu and Menu Item Names" on page 396 for valid values. Or you can specify the value using the constants. The following examples show two ways to specify the Expand All menu item under the Data menu: `name="<%= ModelConstants.DATA_EXPAND_ALL %>"` `name="dataExpandAll"`
accesskey	No	The accelerator key. The specified character is used to search through the title of this menuItem. If it is found, the character is underlined in the menuItem's title in the user interface, indicating that it is the keyboard accelerator for the menuItem.
checkable	No	true to make the menu item checkable. When the menu item is selected, a check mark appears before the menu item. **Note:** If you want to set checkable or checked attributes on built-in menu items, you will need to add your custom event handler and controller, or the settings will have no effect.
checked	No	true to have a check mark appear before the menu item.
disabled	No	Set this to true to disable the menu item. The menu item is displayed in the menu, but is greyed out.

Attribute	Required	Description
imageURL	No	The URL of the image to use. If themeBasedImage is set to true, your custom images need to reside in the directory where the theme's images are stored in the DB2 Alphablox repository. Typically, this directory is located at: <alphablox_dir>/repository/theme/<themeName>/i/ If themeBasedImage is set to false, specify the URL of the image. The URL can be: • An absolute URL. The string should begin with "http://". • A relative URL: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. – Starting the string without a slash indicates that the URL is relative to the current document.
positionBefore	No	The position where the named menu item should be placed. If this is not specified, newly added menu item is appended to the end of the menu.
separator	No	Set to true to add a separator line.
themeBasedImage	No	Set to true to use theme-based images. Images need to reside in the directory where theme's images are stored in the repository. Typically, this directory is located at: <alphablox_dir>/repository/theme/<themeName>/i/ Set to false to use images that do not reside in the theme's image directory. Use the imageURL attribute to specify the URL of the image file.
title	Yes for custom menu items	The displayed title of this menu item. Custom menu items added must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse-over.
visible	No	The visibility of the menu item. When set to false, the menu item is not displayed in the menu. The default is true.

Nested <bloxui:clientLink> Tag Attribute

This is a nested tag for multiple Blox UI tags. See "The <bloxui:clientLink> Tag" on page 412 for details.

Built-in Menu and Menu Item Names

All the common component names used by the Blox UI model are constants. You can find all the constants in the ModelConstants interface under the com.alphablox.blox.uimodel package in the Javadoc documentation. Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with no underscores ("_"), with the first letter of second and each subsequent word in uppercase. The following table for the built-in menu and menu item names is provided for your convenience. For a complete list of model constants, see "Model Constants and Their Values" on page 413.

Menu	Menu Item	Menu Constants
fileMenu		FILE_MENU
	fileOpen	FILE_OPEN
	fileSaveAs	FILE_SAVE_AS
	fileExportToExcel	FILE_EXPORT_TO_EXCEL
	fileExportToPDF	FILE_EXPORT_TO_PDF
editMenu		EDIT_MENU
	editUndo	EDIT_UNDO
	editRedo	EDIT_REDO
	editFind	EDIT_FIND
	editHistory	EDIT_HISTORY
	editCopy	EDIT_COPY
	editDelete	EDIT_DELETE
	editSelectAll	EDIT_SELECTALL
viewMenu		VIEW_MENU
	viewChart	VIEW_CHART
	viewGrid	VIEW_GRID
	viewPageFilter	VIEW_PAGE_FILTER
	viewDataLayout	VIEW_DATA_LAYOUT
	viewToolbarMenu	VIEW_TOOLBAR_MENU
	viewToolbarCustomize	VIEW_TOOLBAR_CCUSTOMIZE
	viewPoppedOut	VIEW_POPPED_OUT
dataMenu		DATA_MENU
	dataSortAscending	DATA_SORT_ASCENDING
	dataSortDescending	DATA_SORT_DESCENDING
	dataDrillUp	DATA_DRILL_UP
	dataDrillDown	DATA_DRILL_DOWN
	dataExpandAll	DATA_EXPAND_ALL
	dataPivot	DATA_PIVOT
	dataShowOnly	DATA_SHOW_ONLY
	dataRemoveOnly	DATA_REMOVE_ONLY
	dataKeepOnly	DATA_KEEP_ONLY
	dataHide	DATA_HIDE
	dataUnhideAll	DATA_UNHIDE_ALL
	dataSwapAxes	DATA_SWAP_AXES
	dataOptions	DATA_OPTIONS
	dataNavigateButton	DATA_NAVIGATION_BUTTON
	dataAdvancedMenu	DATA_ADVANCED_MENU
	dataAdvancedDrillThrough	DATA_ADVANCED_DRILL_THROUGH
	dataAdvancedFormatMask	DATA_ADVANCED_FORMAT_MASK
	dataAdvancedMergedHeaders	DATA_ADVANCED_MERGED_HEADERS
	dataAdvancedSetHiddenColumns	DATA_ADVANCED_SET_HIDDEN_COLUMNS

Menu	Menu Item	Menu Constants
	dataAdvancedSetHiddenMembers	DATA_ADVANCED_SET_HIDDEN_MEMBERS
	dataAdvancedSetHiddenMenu	DATA_ADVANCED_SET_HIDDEN_MENU
	dataAdvancedSetHiddenRows	DATA_ADVANCED_SET_HIDDEN_ROWS
	dataAdvancedShowBottomLevel	DATA_ADVANCED_SHOW_BOTTOM_LEVEL
	dataAdvancedShowSiblings	DATA_ADVANCED_SHOW_SIBLILING
	dataAdvancedTrafficLights	DATA_ADVANCED_TRAFFIC_LIGHTS
	dataCalculationEditor	DATA_CALCULATION_EDITOR
	dataCommentsMenu	DATA_COMMENTS_MENU
	dataCommentsAddComment	DATA_COMMENTS_ADD_COMMENT
	dataCommentsDisplayComments	DATA_COMMENTS_DISPLAY_COMMENTS
	dataMemberFilter	DATA_MEMBER_FILTER
chartMenu		CHART_MENU
	chartTypesMenu	CHART_TYPES_MENU
	chartTypesLine	CHART_TYPES_LINE
	chartTypesBar	CHART_TYPES_BAR
	chartTypes3DBar	CHART_TYPES_3DBAR
	chartTypes3DPie	CHART_TYPES_3DPIE
	chartTypesMore	CHART_TYPES_MORE
	chartAxisPlacement	CHART_AXIS_PLACEMENT
	chartComboTypes	CHART_COMBO_TYPES
	chartDataValues	CHART_DATA_VALUES
	chartAllData	CHART_ALL_DATA
	chartSelectedData	CHART_SELECTED_DATA
	chartOptions	CHART_OPTIONS
toolsMenu		TOOLS_MENU
	toolsGridOptions	TOOLS_GRID_OPTIONS
	toolsPresentOptions	TOOLS_PRESENT_OPTIONS
	toolsManageMenu	TOOLS_MANAGE_MENU
	toolsManageTrafficLights	TOOLS_MANAGE_TRAFFIC_LIGHTS
helpMenu	helpHelp	HELP_HELP
	helpAbout	HELP_ABOUT

Menu Tag Examples

Example 1: Removing a menu item

This example demonstrates how to remove a menu item from the menu bar by setting the visibility of the menu item to `false`. In this example, the Edit menu item and the Grid Options... submenu under Tools are removed.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
```

```
</head>
<body>
...
<blox:present menubarVisible="true" ...>
...
   <bloxui:menu name="editMenu" visible="false" />
   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="toolsGridOptions" visible="false" />
   </bloxui:menu>
...
</blox:present>
...
</body>
</html>
```

Example 2: Disabling a menu item

This example demonstrates how to disable a menu item from the menu bar by setting the `disabled` attribute of the menu item to `true`. In this example, the Grid Options... submenu under Tools is disabled.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<html>
<head>
   <blox:header />
</head>
<body>
...
<blox:present menubarVisible="true"...>
...
   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="toolsGridOptions" disabled="true" />
   </bloxui:menu>
...
</blox:present>
</body>
</html>
```

Example 3: Creating a menu item

This example creates a menu item called “Quick Links” with three options. The second option has submenus.

- When the first option “Today’s Stock Quotes” is selected, a page located on another server is loaded in a separate browser window.
- A separator is added between option 1 and option 2.
- When either of the two submenus in the second option “Reports...” are selected, a page on the same server is loaded in a separate browser window.
- When the third option “Calendar” is selected, a JavaScript function is called.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>
<blox:present menubarVisible="true"...>
...
   <bloxui:menu name="myMenu" title="Quick Links" visible="true">
      <bloxui:menuItem name="option1" title="Today's Stock Quotes">
         <bloxui:clientLink link="http://myserver/quotes.jsp"
```

```
            target="mywindow" />
         </bloxui:menuItem>

         <bloxui:menuItem separator="true" />

         <bloxui:menu name="option2" title="Reports...">
            <bloxui:menuItem name="submenu1" title="YTD Sales- East">
               <bloxui:clientLink link="east.jsp"
                  target="mywindow" />
            </bloxui:menuItem>

            <bloxui:menuItem name="submenu2" title="Google">
               <bloxui:clientLink link="central.jsp"
                  target="myotherwindow" />
            </bloxui:menuItem>
         </bloxui:menu>

         <bloxui:menuItem name="option3" title="Calendar">
            <bloxui:clientLink link="javascript:getCalendar();" />
         </bloxui:menuItem>
      </bloxui:menu>

   </blox:present>

   </body>
   </html>
```

Custom Toolbar Tags

The UI tags for customizing the toolbar allow you to add, remove, or disable menus and menu items to the default toolbar in a PresentBlox, GridBlox, or ChartBlox. Its tags need to be nested inside the tags for these user interface Blox. Whenever possible, however, you should use ToolbarBlox's tag attributes to set its property values. For example, you can use the `removeButton` tag attribute to remove buttons. Use the `<bloxui:toolbar>` and `<bloxui:toolbarButton>` tags only if you are using the DHTML client and if you need higher level of toolbar customization than is provided by Blox properties. As are all the Blox UI tags, these tags use styles that override the theme-based Cascading Stylesheet class settings used in the DHTML client.

This section describes the general concepts related to the `Toolbar` and `ToolbarButton` components and provides tag reference for these components.

Toobar and ToolbarButton

There are two default toolbars in a PresentBlox: Standard and Navigation. Each of the toolbars is a Toolbar component that you can remove, disable, or reposition. Each toolbar contains toolbar buttons. Each toolbar button can also be removed, disabled, or repositioned. In addition, you can add your custom toolbar and toolbar buttons, or customize the operations associated with the toolbar buttons.

When you add a custom toolbar, the menu bar's View -> Toolbar menu option will automatically include your custom toolbar in the list.

Toolbar Tags Listing

```
<bloxui:toolbar
   disabled=""
   bloxRef=""
   name=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible="">
   <bloxui:toolbarButton
      checkable=""
      checked=""
      disable=""
      imageURL=""
      name=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible="" >
      <bloxui:clientLink
         features=""
         link=""
         target="" />
   </bloxui:toolbarButton>
</bloxui:toolbar>
```

The <bloxui:toolbar> Tag Attributes

Attribute	Required	Description
name	Yes	The name of the toolbar. Specify your custom toolbar name to add a custom toolbar. If a Toolbar with the specified name is found, then the tag acts on the component. Otherwise, a new Toolbar is created. To customize the two out-of-the-box toolbars when a presentation Blox is created with its toolbar turned on, specify one of the following values: • Using the constants: – NAVIGATION_TOOLBAR – STANDARD_TOOLBAR • Using valid values: – navigationToolbar – standardToolbar The following examples show two ways to specify the Standard toolbar: `name="<%= ModelConstants.STANDARD_TOOLBAR %>"` `name="standardToolbar"`
bloxRef	No	References an existing Blox to apply the tag to when the tag is used outside of the Blox's tag. See "Example on bloxRef Attribute" on page 371.
disabled	No	Set this to true to disable a toolbar. The toolbar is displayed but clicking the buttons have no effect.

Attribute	Required	Description
positionBefore	No	The position before which the toolbar should be displayed. If this is not specified, newly added toolbar is appended to the end (after the Navigation Toolbar). For example, to position your custom toolbar before the Navigation Toolbar: `positionBefore="navigationToolbar"`
resourceName	No	Loads the named resource file into the component. This allows you to have a toolbar tag that creates a new toolbar from a toolbar XML file. See Chapter 25, "XML Resource Files Reference," on page 421 for more information.
title	Yes for custom toolbar	The displayed title for the toolbar. Custom toolbars added must have a title. This title is displayed under the button icon image when the ToolbarBlox's `textVisible` property is set to true (default is `false`). It is also automatically added to the menu bar's View -> Toolbar menu option in the same nesting presentation Blox (PresentBlox, GridBlox, or ChartBlox) when the Blox's `menubarVisible` property is set to `true`. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse over.
visible	No	The visibility of the toolbar. When set to `false`, the entire toolbar is not displayed. The default is `true`.

The <bloxui:toolbarButton> Tag

Attribute	Required	Description
name	Yes	The name of the toolbar button. Specify your custom toolbar button names to add your custom toolbar button. If a ToolbarButton with the specified name is found, then the tag acts on the component. Otherwise, a new ToolbarButton is created. To customize a built-in toolbar button, see "Built-in Toolbar and ToolbarButton Names" on page 404 for valid values. The following examples show two ways to specify the Copy button in the Standard toolbar: `name="<%= ModelConstants.EDIT_COPY %>"` `name="editCopy"`
checkable	No	`true` to make the toolbar button sticky. That is, the button stays depressed until you click another button. **Note:** If you want to set `checkable` or `checked` attributes on built-in toolbar buttons, you will need to add your custom event handler and controller, or the settings will have no effect.
checked	No	Set this to `true` to have the toolbar button appear to be depressed.
disabled	No	Set this to `true` to disable the button. The button icon is displayed in the toolbar, but does not respond to mouseOver or onClick events. For custom buttons you want to disable, an image of the same name with a suffix of "`_disabled`" should be supplied. See the `imageURL` attribute for more information.

Attribute	Required	Description
imageURL	No	The URL of the image to use. If themeBasedImage is set to true, your custom images need to reside in the directory where the theme's images are stored in the DB2 Alphablox repository. Typically, this directory is located at: <alphablox_dir>/repository/theme/<themeName>/i/ If themeBasedImage is set to false, specify the URL of the image. The URL can be: • An absolute URL. The string should begin with "http://". • A relative URL: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. – Starting the string without a slash indicates that the URL is relative to the current document. **Tip:** For each icon, if the ToolbarBlox's rolloverEnabled property is set to true (the default), you should supply two images, one for non-active mode and the other for active mode. When the toolbar button is selected (active mode), DB2 Alphablox automatically looks for an image of the same name but with a "_active" suffix. If this image file does not exist, the browser will display a missing icon. For any disabled button (button that is displayed but does not respond to mouseOver or onClick events), you should also supply an image with the same name with a "_disabled" suffix.
positionBefore	No	The position where the named toolbar button should be placed. If this is not specified, newly added toolbar button is appended to the end of the toolbar.
separator	No	Set to true to add a separator bar.
themeBasedImage	No	Set to true to use theme-based images. Images need to reside in the directory where theme's images are stored in the repository. Typically, this directory is located at: <alphablox_dir>/repository/theme/<themeName>/i/ Set to false to use images that do not reside in the theme's image directory. Use the imageURL attribute to specify the URL of the image file.
title	Yes for custom menu items	The displayed title of this toolbar button. Custom toolbar button added must have a title. The title cannot contain slashes ("/").
tooltip	No	Tooltip displayed with mouse over.
visible	No	The visibility of the toolbar button. When set to false, the toolbar button is not displayed in the menu. The default is true.

Built-in Toolbar and ToolbarButton Names

All the common component names used by the Blox UI model are constants. You can find all the constants in the `ModelConstants` interface under the com.alphablox.blox.uimodel package in the Javadoc documentation. Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with the first letter of second and each subsequent word in uppercase. The following table for the built-in toolbars and toolbar button names is provided for your convenience. These names should only be used with the `<bloxui:toolbar>` and `<bloxui:toolbarButton>` tags and do not apply to ToolbarBlox's `removeButton` property. For a complete list of model constants, see "Model Constants and Their Values" on page 413.

Toolbar

Buttons in standardToolbar	Buttons in navigationToolbar
`fileOpen`	`dataNavigateButton`
`fileSaveAs`	`dataSortAscending`
`editCopy`	`dataSortDescending`
`standardToolbarSeparator1`(the separator after the editCopy button)	`dataMemberFilter`
`viewPoppedOut`	`navigationToolbarViewSeparator`
`editRedoButton`	`viewGrid`
`editUndoButton`	`viewChart`
`fileExportToPDF`	`viewPageFilter`
`fileExportToExcel`	`viewDataLayout`
`standardToolbarSeparator3`	
`helpHelp`	

The Undo, Redo, and data navigation buttons (which contains various navigation options such as Drill Down, Drill Up, Pivot, and Show Only) are `DropDownToolbarButton` components, not `ToolbarButton` components. However, they can still be removed using the `<bloxui:toolbarButton>` tag:

```
 <bloxui:toolbar name="navigationToolbar" visible="true">
   <bloxui:toolbarButton
      name="<%=ModelConstants.DATA_NAVIGATE_BUTTON%>" visible="false"/>
</bloxui:toolbar>
```

Alternatively, you can also remove these DropDownToolbarButtons using the generic `<bloxui:component>` tag:

```
<bloxui:toolbar name="navigationToolbar" visible="true">
   <bloxui:component name="<%=ModelConstants.DATA_NAVIGATE_BUTTON%>"
      visible="false"/>
</bloxui:toolbar>
```

Note: The setting of `maximumUndoSteps`, a common Blox property controls the availability of the Undo and Redo buttons. If `maximumUndoSteps` is set to 0, then the Undo and Redo buttons will be removed. If `maximumUndoSteps` is not 0, these buttons will show. See "maximumUndoSteps" on page 36

Toolbar Tags Examples

Example 1: Removing a toolbar buttons

This example demonstrates how to remove a toolbar button from the Navigation toolbar by setting the visibility of the toolbar button to `false`.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>

<blox:present ....>
...
<bloxui:toolbar name="navigationToolbar" >
   <bloxui:toolbarButton name="viewGrid" visible="false" />
   <bloxui:toolbarButton name="viewChart" visible="false" />
   <bloxui:toolbarButton name="viewPageFilter" visible="false" />
   <bloxui:toolbarButton name="viewDataLayout" visible="false" />
</bloxui:toolbar>
...
</blox:present>
</body>
</html>
```

Example 2: Adding a custom toolbar

This example creates a Toolbar called "myToolbar" (`name="myToolbar"`) with a display name of "My Toolbar" (`title="My Toolbar"`).

This example demonstrates:

- the use of the `positionBefore` attribute to specify the toolbar position.
- the use of absolute and relative URLs to specify the image URLs.
- the use of the `<bloxui:clientLink>` nested tag to specify a URL when the toolbar button is clicked (see "The <bloxui:clientLink> Tag" on page 412 for more information).

The menu bar will automatically reflect this new toolbar in its View -> Toolbar... menu option.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>

<body>
<blox:present id="myPresentBlox" width="700" height="500" >
   <blox:data dataSourceName="TBC" useAliases="true"
      query="<SYM <ROW(Product) <ICHILD Product <COLUMN(Year,
         Scenario) Qtr1 Qtr2 <CHILD Scenario Sales !" />
   <bloxui:toolbar name="myToolbar" title="My Toolbar"
      visible="true" positionBefore="navigationToolbar">

      <bloxui:toolbarButton name="option1" title="mail"
         themeBasedImage="false"
         imageURL="http://myserver/myApp/email.gif"
         tooltip="Check email alerts">
         <bloxui:clientLink link="emailAlerts.jsp"
            target="mywindow"
            features="toolbar=no,status=no" />
      </bloxui:toolbarButton>
```

```
      <bloxui:toolbarButton name="option2" title="Stocks"
         themeBasedImage="false" imageURL="../money.gif"
         tooltip="Today's Stocks">
         <bloxui:clientLink link="http://www.my.com/app/file.jsp"
            target="mywindow" />
      </bloxui:toolbarButton>

      <bloxui:toolbarButton name="option3" title="KPI"
         themeBasedImage="false" imageURL="/myApp/lookup.gif"
         tooltip="Show KPI" >
         <bloxui:clientLink link="javascript:myLookupFunction()"
            target="mywindow" />
      </bloxui:toolbarButton>

      <bloxui:toolbarButton separator="true"
         positionBefore="option1" />
   </bloxui:toolbar>

</blox:present>
</body>
</html>
```

Accessibility Tag

The <bloxui:accessibility> tag is designed to enhance the user experience for users with disabilities.

- It offers a screenReaderMode attribute for allowing users with limited vision to go right to the data contents of a Blox without having to tab through the menu bar or toolbars. For example, when the <bloxui:accessibility> tag is added to a PresentBlox and the screenReaderMode attribute is set to true, three links are added before the rendered PresentBlox for access to the nested GridBlox, ChartBlox, and DataLayoutBlox. When users tab to links, the screen reader will read the text of the links, indicating to them that they can press Enter to go to the subcomponents directly.
- It has a windowMenuEnabled attribute for providing keyboard access to open dialogs that have lost focus. When the windowMenuEnabled attribute is set to true, a **Windows** menu is added to the menu bar that contains a menu item for each open dialog. In the unlikely events an open dialog on the page is no longer in focus and keyboard users cannot tab to the dialog, they can tab to the menu bar and the specific menu item to bring the dialog into focus.

The <bloxui:accessibility> Tag

This tag has the following attributes:

```
<bloxui:accessibility
   screenReaderMode=""
   windowMenuEnabled=""
 />
```

where:

Attribute	Required	Default	Description
screenReaderMode	No	false	Adds a link before the rendered Blox component to each major subcomponent. If this is a PresentBlox, three links are added for the nested GridBlox, ChartBlox, and DataLayoutBlox. Only PresentBlox, GridBlox, ChartBlox and DataLayoutBlox are affected by the presence of this attribute.
windowMenuEnabled	No	false	Adds a **Windows** menu to the menu bar that contains a menu item for each open dialog. Selecting a menu item for an open dialog allows users to bring that dialog into focus using only the keyboard. It also adds a **Close All Dialogs** menu item for closing all dialogs. PresentBlox, standalone GridBlox, and standalone ChartBlox are affected by the presence of this attribute.

Example

The following example adds a **Windows** menu to the menu bar and links before a PresentBlox for quick access to the nested GridBlox, ChartBlox, and DataLayoutBlox.

```
<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
...
<blox:present id="accessiblePresent" menubarVisible="true">
   <blox:toolbar visible="true" />
   <blox:data bloxRef="myData" />
   <bloxui:accessibility
      screenReaderMode="true"
      windowMenuEnabled="true"
   />
</blox:present>
```

Utility Tags

The Blox UI Tag Library contains a set of utility tags. These tags allow you to specify actions to take when a ClickEvent is triggered in a UI component, set properties on classes referenced by the <bloxui:customLayout> and <bloxui:customAnalysis> tags, or to invoke server-side code to customize GridBlox layout. They include the following tags:

The <bloxui:actionFilter> Tag

The <bloxui:actionFilter> tag allows you to invoke server-side code from a Blox UI component when it is clicked using the Blox UI Tag Library.

```
<bloxui:actionFilter
   componentName=""
   filter="" />
```

where:

Attribute	Description
componentName	The name of the component this action filter should be attached to. When this named component is clicked, a ClickEvent is triggered and the action specified in the actionFilter method of the named class is performed.
filter	Specifies the filter object. For example: `<bloxui:actionFilter` `filter="<%= new MyActionFilterClass() %>"` `componentName="dataPivot" />` where MyActionFilterClass implements IActionFilter and is defined within the JSP page.

The IActionFilter interface in the com.alphablox.blox.uimodel.tags package must be implemented by all action filters added to Blox using the <bloxui:actionFilter> tag. This interface has one method with the following signature:

```
void actionFilter(DataViewBlox blox, Component component);
    //throws java.lang.Exception
```

where:

Argument	Description
blox	The action filter's Blox
component	The component that generates the ClickEvent

This method is called each time the component this action filter is attached to generate a ClickEvent. You can implement this method and add actions to take when the associated component is clicked.

In order to implement this method, make sure at least the following packages are imported:

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.DataViewBlox,
                  com.alphablox.blox.uimodel.core.Component" %>
```

You may also need to import other packages. Use a Java Integrated Development Environment (IDE) to help you identify which packages to import.

Utility Tags Example

The following example demonstrates the use of the <bloxui:actionFilter> tag and how to implement the IActionFilter interface and to extend its actionFilter method to take some action when a component is clicked. In this case, when the custom menu item is clicked, a MessageBox is displayed with some message.

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.DataViewBlox,
```

```
                com.alphablox.blox.uimodel.core.Component,
                com.alphablox.blox.uimodel.core.MessageBox"%>

<%@ taglib uri="bloxtld" prefix="blox"%>
<%@ taglib uri="bloxuitld" prefix="bloxui"%>

<html>
<head>
   <blox:header />
</head>

<blox:present ....>

   <bloxui:menu name="toolsMenu" >
      <bloxui:menuItem name="myToolMenuItem" title="Get Message" />
   </bloxui:menu>

   <bloxui:actionFilter
      filter="<%= new MyActionFilterClass() %>"
      componentName="myToolMenuItem" />
...
</blox:present>
...
<%!
public static class MyActionFilterClass implements IActionFilter
{
   public void actionFilter( DataViewBlox blox, Component component )
throws Exception {
      MessageBox.message( component, "Get Message", "The myToolMenuItem has
been clicked!" );
   }
}
%>
```

The <bloxui:gridFilter> Tag

The <bloxui:gridFilter> tag allows you to invoke server-side code and customize GridBlox layout using the Blox UI Tag Library. It has the following tag attributes:

```
<bloxui:gridFilter
   filter="" />
```

where:

Attribute	Description
filter	Specifies the filter object. For example: `<bloxui:gridFilter` `filter="<%= new MyGridFilterClass() %>"` `componentName="dataPivot" />` where MyGridFilterClass implements IGridFilter and is defined within the JSP page.

You can use the grid filters to customize and rebuild a grid before it is loaded. The IGridFilter interface in the com.alphablox.blox.uimodel.tags package must be implemented by all grid filters added to Blox using the <bloxui:gridFilter> tag. The grid is rebuilt after each data navigation command is processed. This interface has two methods with the following signature:

```
void gridFilter(DataViewBlox blox, GridBrixModel grid);
                  // throws java.lang.Exception
void cellFilter(DataViewBlox blox, GridCell cell);
                  // throws java.lang.Exception
```

where:

Argument	Description
`blox`	The grid filter's Blox
`grid`	The grid which has been rebuilt
`cell`	The grid cell which has been rebuilt

Since this filter may be one of many filters that are applied to the grid after a rebuild, the filter should not make assumptions regarding the layout of the grid.

In order to implement this method, make sure at least the following packages are imported:

```
<%@ page import="com.alphablox.blox.uimodel.*,
                 com.alphablox.blox.uimodel.tags.IActionFilter,
                 com.alphablox.blox.uimodel.tags.IGridFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.GridBrixModel,
                 com.alphablox.blox.uimodel.GridBrixCellModel,
                 com.alphablox.blox.uimodel.core.grid.GridRow,
                 com.alphablox.blox.uimodel.core.grid.GridCell,
                 com.alphablox.blox.uimodel.core.Component" %>
```

You may also need to import other packages. Use a Java Integrated Development Environment (IDE) to help you identify which packages to import.

gridFilter Tag Example

The following example demonstrates the use of the `<bloxui:gridFilter>` tag and how to implement the `IGridFilter` interface and to extend its `gridFilter` method to rebuild a grid. In this case,

- We move
  - The row headers are moved to the end of the grid.
  - The column headers are moved to the bottom of the grid.
- A Button component is added to the end of each column.
- Once the grid is rebuilt, pop up a MessageBox notifying that the grid has changed.
- The grid is then displayed.

See the Blox Sampler for a similar example.

```
<%@ page import="com.alphablox.blox.uimodel.tags.IGridFilter,
                 com.alphablox.blox.DataViewBlox,
                 com.alphablox.blox.uimodel.GridBrixModel,
                 com.alphablox.blox.uimodel.core.grid.GridRow,
                 com.alphablox.blox.uimodel.core.grid.GridCell,
                 com.alphablox.blox.uimodel.core.Button,
                 com.alphablox.blox.uimodel.ModelConstants,
                 com.alphablox.blox.uimodel.core.grid.GridColumn,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.GridBrixCellModel"%>
<%@ taglib uri="bloxtld" prefix="blox" %>
<%@ taglib uri="bloxuitld" prefix="bloxui" %>

<blox:data id="dataBlox" dataSourceName="qcc-essbase"
   useAliases="true" visible="false"
   query="<ROW ('All Locations') 'Central' 'East' 'West' 'All Locations'
<ASYM <COLUMN ('Scenario', 'All Time Periods') 'Actual' 'Actual' 'Forecast'
'Forecast' '2000.Q3' '2000.Q4'  '2001.Q1'  '2001.Q2'!" />
```

```
<html>
<head>
   <blox:header />
</head>

<body>
<blox:grid id="testGridMoveFilter" width="80%" height="500"
   bandingEnabled="true" menubarVisible="true">
   <blox:toolbar visible="true" />
   <bloxui:gridFilter
      filter="<%= new MyGridFilterClass() %>" />
   <blox:data bloxRef="dataBlox" />
</blox:grid>

</body>
</html>

<%!
public static class MyGridFilterClass implements IGridFilter
{
   public void gridFilter( DataViewBlox blox, GridBrixModel grid ) throws
Exception {
      // Move row headers to the end of the grid
      if ( grid.getColumnCount() > 1 )
         while ( grid.getColumn( 0 ).isHeader() )
            grid.moveColumn( 0, grid.getColumnCount() );

      // Move column headers to the end of the grid
      if ( grid.getRowCount() > 1 )
         while ( grid.getRow( 0 ).isHeader() )
            grid.moveRow( 0, grid.getRowCount() );

      GridRow row = new GridRow( );

      // Add a button to the end of each column. For this example,
      // these buttons do not do anything.
      for ( int i=0; i < grid.getColumnCount(); i++ ) {
         GridCell cell = new GridCell( "myCell" + (i+1)  );
         cell.add( new Button( cell.getName(), cell.getName() ) );
         cell.setClickable( false );
         row.add( cell );
      }

      grid.addRow( row );

      row = new GridRow();
      row.setHeight( 4 );
      row.setThemeClass( ModelConstants.THEME_STYLE_ROW_DATA_GENERATION +
"3" );
      grid.insertRow( 4, row );

      GridColumn column = new GridColumn();
      column.setWidth( 4 );
      column.setThemeClass( ModelConstants.THEME_STYLE_ROW_DATA_GENERATION
+ "3" );
      grid.insertColumn( 4, column );
      MessageBox.message( grid, "Change", "The grid has changed" );
   }
   public void cellFilter( DataViewBlox blox, GridCell cell ) throws
Exception {
   }
}
%>
```

Note that this example only serves to demonstrate how to customize and rebuild the grid layout with the <bloxui:gridFilter> tag. This is an advanced technique

and may impact how the grid scrolls. In addition, the buttons added to the grid do not have an associated action defined when they are clicked.

The <bloxui:clientLink> Tag

The <bloxui:clientLink> tag allows you to specify an URL to load into the existing or a different browser window once a component is clicked. It should be added within a component tag such as <bloxui:menuItem> and <bloxui:toolbarButton>. It has the following tag attributes.

```
<bloxui:clientLink
   features=""
   link=""
   target="" />
```

Attribute	Description
features	A comma-separated browser feature string. When the target attribute is specified, this attribute sets the features for the new browser window. For example, features="toolbar=no,status=no". The browser feature string should be specified the same way as the JavaScript's window.open() method.
link	An URL. This can be one of the following formats: • An absolute URL. The string should begin with "http://". • A relative URL: – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. – Starting the string without a slash indicates that the URL is relative to the current document. • a JavaScript function name. The string should begin with a "javascript:" prefix.
target	The name of the browser window to load the URL specified in the link attribute with the browser features specified in the features attribute. If this attribute is not specified, the URL is loaded into the current browser window.

See "Example 3: Creating a menu item" on page 399 and "Example 2: Adding a custom toolbar" on page 405 for examples on how this tag is used in conjunction with the other Blox UI tags.

The <bloxui:setProperty> Tag

The <bloxui:setProperty> tag allows you to set the value of a property for a layout or analysis class. For example, when you specify to use a class in your <bloxui:customAnalysis> or <bloxui:customLayout> tag, you can set the value for a named property using this <bloxui:setProperty> tag. It has the following tag attributes:

```
<bloxui:setProperty
   name=""
   value="" />
```

where:

Attribute	Description
name	The name of the property.
value	The value of the property.

setPropertyTag Example

The following example shows how to set the default number of members to show in the popup dialog triggered by the TopN analysis class:

```
<bloxui:customAnalysis
   analysis="<%= new TopN()" >
   <bloxui:setProperty name="number" value="15" />
</bloxui:customAnalysis>
```

See "The <bloxui:customAnalysis> Tag" on page 376 for a complete example.

Model Constants and Their Values

This section lists the common model constants and their values. For a complete list, see the `ModelConstants` interface in the `com.alphablox.blox.uimodel` package in the Javadoc documentation. *Names for the constants are all in uppercase. Their values, when specified in your Blox UI tag attributes, should all be in lowercase with no underscores ("_"), with the first letter of second and each subsequent word in uppercase.*

Chart Elements

Constant	Value
CHART	chart
CHART_FILTER	chartFilter
CHART_FILTERS_CONTAINER	chartFiltersContainer
CHART_TOTALS_FILTER	chartTotalsFilter

Menus

Constant	Value
MAIN_MENU	mainMenu
HELP_MENU	helpMenu
TOOLS_MENU	toolsMenu
DATA_MENU	dataMenu
DATA_ADVANCED_MENU	dataAdvancedMenu
FORMAT_MENU	formatMenu
BOOKMARK_MENU	bookmarkMenu

Constant	Value
VIEW_MENU	viewMenu
CHART_MENU	chartMenu
FILE_MENU	fileMenu
EDIT_MENU	editMenu
VIEW_TOOLBAR_MENU	viewToolbarMenu
CHART_TYPES_MENU	chartTypesMenu
DATA_COMMENTS_MENU	dataCommentsMenu
FORMAT_LAYOUT_MENU	formatLayoutMenu
TOOLS_MANAGE_MENU	toolsManageMenu

Menus Elements

Constant	Value
BOOKMARK_ADD	bookmarkAdd
BOOKMARK_LOAD	bookmarkLoad
BOOKMARK_ORGANIZE	bookmarkOrganize
CHART_COMBO_TYPES	chartComboTypes
CHART_AXIS_PLACEMENT	chartAxisPlacement
CHART_DATA_VALUES	chartDataValues
CHART_ALL_DATA	chartAllData
CHART_SELECTED_DATA_ONLY	chartSelectedDataOnly
CHART_OPTIONS	chartOptions
CHART_TYPES_PIE	chartTypePie
CHART_TYPES_LINE	chartTypeLine
CHART_TYPES_BAR	chartTypesBar
CHART_TYPES_MENU	chartTypesMenu
CHART_TYPES_3DPIE	chartType3DPie
CHART_TYPES_MORE	chartTypesMore
CHART_TRENDLINES	chartTrendlines
HELP_HELP	helpHelp
HELP_ABOUT	helpAbout
DATA_SORT	dataSort
DATA_SORT_ASCENDING	dataSortAscending
DATA_SORT_DESCENDING	dataSortDescending
DATA_DRILL_UP	dataDrillUp
DATA_DRILL_DOWN	dataDrillDown
DATA_EXPAND	dataExpand
DATA_COLLAPSE	dataCollapse
DATA_EXPAND_ALL	dataExpandAll
DATA_PIVOT	dataPivot
DATA_SHOW_ONLY	dataShowOnly

Constant	Value
DATA_REMOVE_ONLY	dataRemoveOnly
DATA_KEEP_ONLY	dataKeepOnly
DATA_HIDE	dataHide
DATA_UNHIDE_ALL	dataUnhideAll
DATA_SWAP_AXES	dataSwapAxes
DATA_MEMBER_FILTER	dataMemberFilter
DATA_CALCULATION_EDITOR	dataCalculationEditor
DATA_OPTIONS	dataOptions
DATA_ADVANCED_DRILL_THROUGH	dataAdvancedDrillThrough
DATA_ADVANCED_FORMAT_MASK	dataAdvancedFormatMask
DATA_ADVANCED_MERGED_HEADERS	dataAdvancedMergedHeaders
DATA_ADVANCED_SHOW_BOTTOM_LEVEL	dataAdvancedShowBottomLevel
DATA_ADVANCED_SHOW_SIBLINGS	dataAdvancedShowSiblings
DATA_ADVANCED_SET_HIDDEN_MENU	dataAdvancedSetHidden
DATA_ADVANCED_SET_HIDDEN_ROWS	dataAdvancedSetHiddenRows
DATA_ADVANCED_SET_HIDDEN_COLUMNS	dataAdvancedSetHiddenColumns
DATA_ADVANCED_SET_HIDDEN_MEMBERS	dataAdvancedSetHiddenMembers
DATA_ADVANCED_TRAFFIC_LIGHTS	dataAdvancedTrafficLights
DATA_COMMENTS_DISPLAY_COMMENTS	dataCommentsDisplayComments
DATA_COMMENTS_ADD_COMMENT	dataCommentsAddComment
EDIT_UNDO	editUndo
EDIT_REDO	editRedo
EDIT_COPY	editCopy
EDIT_DELETE	editDelete
EDIT_SELECT_ALL	editSelectAll
EDIT_FIND	editFind
EDIT_HISTORY	editHistory
FILE_OPEN	fileOpen
FILE_SAVE_AS	fileSaveAs
FILE_EXPORT_TO_PDF	fileExportToPDF
FILE_EXPORT_TO_EXCEL	fileExportToExcel
TOOLS_GRID_OPTIONS	toolsGridOptions
TOOLS_MANAGE_TRAFFIC_LIGHTS	toolsManageTrafficLights
TOOLS_PRESENT_OPTIONS	toolsPresentOptions
VIEW_GRID	viewGrid
VIEW_CHART	viewChart
VIEW_PAGE_FILTER	viewPageFilter
VIEW_DATA_LAYOUT	viewDataLayout
VIEW_POPPED_OUT	viewPoppedOut
VIEW_TOOLBAR_CUSTOMIZE	viewToolbarCustomize
LOGO	logo

Constant	Value
DATA_NAVIGATE_BUTTON	dataNavigateButton
EDIT_UNDO_BUTTON	editUndoButton
EDIT_REDO_BUTTON	editRedoButton

Dialog Buttons

Constant	Value
OK	ok
CANCEL	cancel
YES	yes
NO	no
APPLY	apply
HELP	help

Toolbars

Constant	Value
STANDARD_TOOLBAR	standardToolbar
NAVIGATION_TOOLBAR	navigationToolbar

General Elements

Constant	Value
HEADER_CONTAINER	headerContainer
BODY_CONTAINER	bodyContainer
PRESENT_SPLITTER	presentSplitter
TREENODE_LABEL	treeNodeLabel
GRID_CELL_VALUE	gridCellValue
DATA_LAYOUT_LIST	dataLayoutList
DATA_LAYOUT_TREE	dataLayoutTree
DATA_LAYOUT_ROW_CONTAINER	dataLayoutRowContainer
DATA_LAYOUT_COLUMN_CONTAINER	dataLayoutColumnContainer
DATA_LAYOUT_PAGE_CONTAINER	dataLayoutPageContainer
DATA_LAYOUT_OTHER_CONTAINER	dataLayoutOtherContainer

Chapter 24. PDF Rendering Tags

The default toolbar in the Blox user interface contains an Export to PDF icon, allowing users to convert their current view of Blox on the page to PDF format for printing or archive. As a developer, you can use the `<blox:pdfReport>` tag to specify the header, footer, their heights, page margin, and page size. In addition, you can customize the popup dialog using the `<blox:pdfDialogInput>` tag to prompt users to specify these various settings. You can also use the provided tags to render multiple Blox on the page to one PDF.

Tags for Rendering to PDF

Below are the tags and attributes associated with PDF rendering:

```
<blox:pdfReport
    header=""
    headerHeight=""
    footer=""
    footerHeight=""
    margin=""
    pageBreak=""
    size=""
    theme=""
    themeListEnabled="" >
    <blox:pdfDialogInput
        index=""
        displayName=""
        defaultValue=""
    />
</blox:pdfReport>
```

For a detailed discussion of PDF report generation and the use of the tags, see the topic on Converting to PDF in the *Developer's Guide*.

pdfReport Tag Attributes

Attributes for `pdfReport`	Description
`header`	The header, including text and layout. Defined using XHTML and macros. See "Macros for PDF Conversion" on page 418.
`headerHeight`	Header height. Valid units include: pixels (`px`), points (`pt`), inches (`in`), millimeters (`mm`), and centimeters (`cm`). If not specified, pixels (`px`) will be used.
`footer`	The footer, including text and layout. Defined using XHTML tags and macros. See "Macros for PDF Conversion" on page 418.
`footerHeight`	Footer height. Valid units include: pixels (`px`), points (`pt`), inches (`in`), millimeters (`mm`), and centimeters (`cm`). If not specified, pixels (`px`) will be used.
`margin`	Margin. Valid units include: pixels (`px`), points (`pt`), inches (`in`), millimeters (`mm`), and centimeters (`cm`). If not specified, pixels (`px`) will be used.

Attributes for pdfReport	Description
pageBreak	Rule for setting the page break. The rule consists of a list of dimension-member specifications, followed by the `@ before` or `@ after` keyword. For example, `Dim1: M1, M2; Dim2: M3, M4 @ after` The above example specifies to add a page break **after** the data whenever the members M1 or M2 of dimension Dim1, or members M3 or M4 of dimension Dim2 change. Separate each dimension-member specification with a semicolon. Separate the members from the same dimension with a comma. A page break is added when any of the specified condition is met. The keywords `@ before` and `@ after` are case-insensitive. If the keyword is spelled incorrectly, an exception is thrown.
size	Paper size, used to define paper size (`A3`, `A4`, `Letter`, `Legal`) and orientation (`landscape`, `portrait`). Valid attributes are: `[ A3 | A4 | Letter | Legal | Custom` `[ [Portrait | Landscape] | [`*`width`* `| [ `*`height`*`]]]` Examples: `size="Letter Portrait"` `size="A4 Landscape"` `size="Custom 15in 100mm"` Default page size is locale-specific. In US or Canada, default is `Letter`, otherwise the page size default will be `A4`. Default orientation is `Landscape`.
theme	Theme name, same as theme name used in DB2 Alphablox Repository.
themeListEnabled	Theme list enabled. Value can be `true` (default) or `false`.

Note: XHTML tag and CSS limitations:

- `<center>` is not supported
- For a non-breaking space, use the Unicode character ( ) instead of the XHTML character ( ).
- CSS shorthand attributes should follow CSS specifications.

Macros for PDF Conversion

Macros available for use in the header and footer tag attributes are:

Macro	Description
`<date/>`	Today's date
`<time/>`	Current time
`<totalpages/>`	Page count
`<pagenumber/>`	Current page
`<pdfDialogInput`*N*`/>`	PDF dialog input text, where N is an integer from 1 to 5. By default, `<pdfdialogINput1/>` defines the header and `<pdfDialogInput2/>` defines the footer.

A pdfReport Tag Example

```
<%
  String footer="<span style='color:red'>Total: <totalpages/> pages</span>";
%
<blox:pdfReport
    footer="<%=footer>"
    footerHeight="20px"
    size="Letter portrait"
    margin="0"
    theme="financial"
    themeListEnabled="false"/>
```

pdfDialogInput Tag Attributes

Attributes for **pdfDialogInput**	Description
index	The header, including text and layout. Defined using XHTML and macros.
defaultValue	Header height. Valid units include: pixels (px), points (pt), inches (in), millimeters (mm), and centimeters (cm). If not specified, pixels (px) will be used.
displayName	The footer, including text and layout. Defined using XHTML tags and macros

The pdfDialogInput tag lets you specify settings that you want users to provide values for. For example:

```
<blox:pdfReport>
   <blox:pdfDialogInput
      displayName="Report header"
      defaultValue="Enter your header here"
      index="1" />
   <blox:pdfDialogInput
      displayName="Report footer"
      defaultValue="Enter your footer here"
      index="2" />
</blox:pdfReport>
```

Your users will be prompted with a dialog that allows them to specify the report header and footer.

Chapter 25. XML Resource Files Reference

This chapter provides general reference for writing XML resource files used to create Blox UI model containers. These XML files allow you to create model containers such as dialogs, toolbars, menus and menu bars using the predefined elements and attributes. You can then write your own controller to control these components.

Resource Files Overview

Resource files are language localizable descriptions of compound model containers. They are standard XML files with predefined elements and attributes. The Blox UI Model reads the resource files and constructs all of the listed Java model objects and sets the specified attributes on each object. These resource files are how the user interface is built in the DHTML client.

You can write your own XML resource files using the formats described in this section. You can add the model UI components listed under the `com.alphablox.blox.uimodel.core` package in an XML resource file and have DB2 Alphablox construct all the Java model objects that can be further extended or modified in code as needed. Typically, you need to attach a controller for models that are created from a resource file to handle updates to and events from a model component.

Resource files you created can be loaded in various ways, such as by putting them in some common server location, or loading them from the application directory. Typically this is done by setting the class path. For instructions on how to set your class path, see the *Administrator's Guide*, under the topic on extending DB2 Alphablox. For a complete example that demonstrates how to write a controller to launch a custom dialog created using an XML resource file, see the topic on Dialogs in the DHTML Client UI Extensibility section in the *Developer's Guide*

The Top-Level Elements

Only one top-level container can be defined per resource file. The top-level element has to be one of the following:

- Dialog
- Menubar
- Menu
- Toolbar
- ComponentContainer

The UI created this way is usually either a dialog box, or menu bar, a menu (such as the right-click menu), or a toolbar. These component containers then contain other components. For example, a dialog box may contain some texts (`Static`), an edit text box (`Edit`), a few check boxes (`CheckBox`), a set of radio buttons (`RadioButton`), and an OK button and a Cancel button (`Button`). Besides being a

top-level element, the ComponentContainer element is often used to group a set of elements for better manipulation of the layout and style.

All the visual UI components descend from the Component base class and are arranged in a hierarchy that provides both formatting control as well as a way to centrally manage sets of primitive components. In an XML resource file, the ComponentContainer element is often used in other component containers to "group" several components. This allows you to better manipulate the layout and set attributes for all the elements in the same ComponentContainer.

For more information on the Blox UI Model, see "Blox UI Model" on page 10.

Supported Argument Types

Only the following argument types are supported for models created using an XML resource file:

- string
- boolean
- integer
- Layout (horizontal, vertical, or grid)
- Style
- alignment

The supported argument types are related to the attribute types as these have corresponding "setter" methods on the Component. Only setters that take these types can be used in the XML file. For information on attributes, see "Attributes" on page 425.

Caching of Resource Files

Model resource files are cached by default. After the initial request to load a resource, resource file changes will require restarting the DB2 Alphablox. To disable caching, place a cache="false" attribute on the top-level resource element and restart the server if it has already started. For example;

```
<Dialog name="myDialog" title="My Own Dialog"
   cache="false" modal="true"
   height="420" width="450" layout="vertical">
   <!--other components omitted -->
</Dialog>
```

Localization

Resource files follow the same localization naming convention as resource bundles. The specified locale's language code is appended to the resource filename before it is loaded. For example, if the locale is set to French, the resource filename should have an appended _fr suffix. If the resource file does not exist, the unmodified resource filename is used.

Elements for XML Resource File

All the model UI components in com.alphablox.blox.uimodel.core package are elements you can add to a resource file. Each element has a set of attributes you can specify. As these UI components inherit the same set of properties from the Component class, these elements have similar attributes you can specify. The

common ones include `name`, `title`, `alignment`, `valignment`, `height`, `width`, `layout`, and more. For a list of these common attributes, see "Common Attributes to All UI Elements" on page 429.

List of Elements

The element names for the UI components have the same names as the classes in the `com.alphablox.blox.uimodel.core` package, with an uppercase first letter for each word in the names. Below is a list of the elements:

Component	Description
`Button`	Push button component.
`CheckBox`	CheckBox component.
`ComponentContainer`	Generic container for UI model objects
`ControlbarContainer`	Container for Controlbars, the base class for control bars (menus and toolbars) that can be contained in a ControlbarContainer.
`Dialog`	Dialog component. Dialogs are floating containers that are used to collect input from and/or show status to users. Create a dialog and then populate the dialog with components such as Buttons, CheckBoxes and RadioButtons to present users with option lists or to make a decision.
`DropDownList`	Drop down list component. A DropDownList consists of a single displayed option with a mechanism to select from a list of other choices. Only one choice may be selected at a time. Use a DropDownList when space is limited and the constant display of possible choices is not required.
`DropDownToolbarButton`	Drop down toolbar button component. The DropDownToolbarButton has both a drop down list of selections as well as an action button to invoke the currently displayed drop down list. The control generates a ClickEvent when the selection is changed or the action button is clicked.
`Edit`	Edit field (text field) component. An Edit component allows the user to enter and modify one or more lines of text. Text can be copied, moved, and inserted into the Edit field using the standard user UI mechanisms.
`GroupBox`	GroupBox component for providing a titled container for dialogs and other models. The GroupBox component is primarily used to group components in dialog boxes. For example, if there are a number of components dedicated to setting options for a chart, then these should be grouped together in a GroupBox with the title "Chart Options." RadioButton components will behave differently inside of a GroupBox if they are not named. All unnamed RadioButtons in a named group will

	become automatically grouped. Pressing one radio button will unselect others in the group.
`Image`	Image component for displaying GIF, JPEG, or other compatible image. Unlike StaticImage, an Image component will generate a ClickEvent when clicked.
`ListBox`	ListBox component.
`Menu`	Menu component consisting of MenuItems and other Menus. Menus inside of Menus will be treated as submenus with the appropriate submenu behavior. MenuItems will display as selections and will generate a ClickEvent when selected. By default, a MenuItem's name is used to construct a handler method in controllers. For example, a MenuItem with the name "drillDown" will map to a method called "actionDrillDown" in controllers. All new menus are assigned a vertical layout by default.
`Menubar`	Menubar component used in conjunction with ControlbarContainer to display top level menus.
`MenuItem`	MenuItem component. This is a selectable item in Menu.
`RadioButton`	RadioButton component.
`Spacer`	Spacer component for adding fixed height and width spacing among components
`SpinnerButton`	Spinner component that accepts integer input from the user and provides buttons to increase/decrease the value. You can set the initial value, increment, and low and high values.
`SplitterContainer`	SplitterContainer component for displaying two components with a splitter bar between them that the user can adjust. Use either the `HorizontalLayout` or the `VerticalLayout` to control the orientation of the splitter.
`Static`	Static component for displaying simple static text such as instructions, labels, and values where interaction is not needed.
`StaticImage`	Component to render a static image which does not respond to user input.
`TabbedContainer`	Container window with tabs for all child containers. This container is used to display a list of tabs corresponding to all child containers. A typical use is to implement a tabbed dialog box. This container can only contain other component containers. The style attached to this container is applied to the tabs. The title of the child container is used for that container's tab label. The selection state of the child containers is used to determine the selected tab. If

	no child containers are selected, then the first container is automatically selected. If multiple child containers are marked as selected, then the first one encountered is considered selected. The order in which the child containers are added to the tabbed container determines the tab order. For top and bottom (horizontal) layout, the first container is on the right. For left and right (vertical) layout, the first container is on the top.
`Toolbar`	Toolbar component used in conjunction with ControlbarContainer to display toolbars.
`ToolbarButton`	ToolbarButton component; can be used anywhere in the component model but are primary designed to work in ControlbarContainers. The `name` of the component is used to construct the image name by appending a `".gif"` extension.

Beside the elements listed above, there are `Item` and `ClientLink`, which are described next.

The Item Element

In the XML resource file, when a `ListBox`, `DropDownList`, or `DropDownToolbarButton` element is added, use the `Item` element to specify the individual items:

```
<DropDownList name="selectList"
      title="Undo"
      tooltip="Select an option" >
      <Item value="A" />
      <Item value="B" />
      <Item value="C" />
</DropDownList>
```

The ClientLink Element

The `ClientLink` element defines a URL-based link that will be handled by the browser when the component is clicked. This element allows you to specify a link, the target window in which the new page is to be loaded, and a browser window feature string (for example, `"toolbar=no,scrollbars=yes"`) in much the same way as the JavaScript's `window.open()` method.

Attributes

Attribute names have lowercase first word, with the first letter in each subsequent word in uppercase, such as `name`, `title`, `width`, `height`, `themeClass`, `imageURL`, and `themeBasedImage`. Since all the UI components derive from the `Component` class, they share many common attributes. These attributes correspond to the "setter" methods on the Component object (for example, `setName()`, `setTitle()`, `setWidth()`, and `setHeight()`). For a listing of these common attributes, see "Common Attributes to All UI Elements" on page 429. For details on their methods, see the Javadoc documentation under the `com.alphablox.blox.uimodel.core` package.

The following example of a simple dialog demonstrates how to add different elements, specify their attributes, and manipulate the layout to create a dialog box using the Blox UI model components.

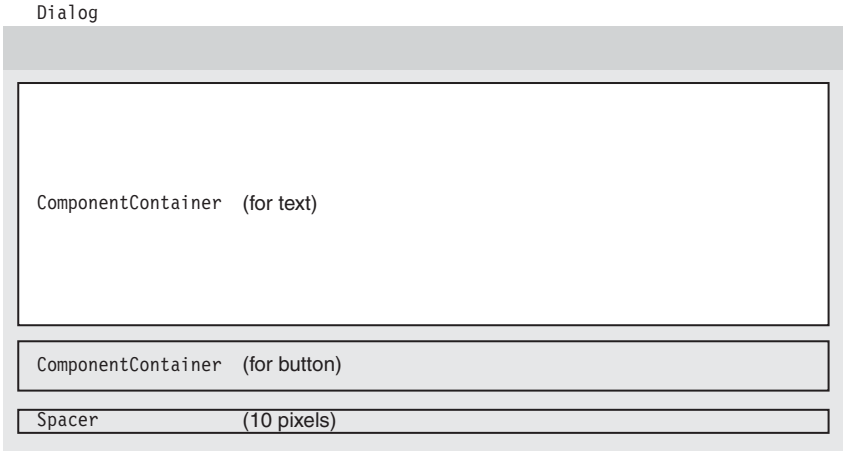
Examples of Resource XML Files

Example 1: An "About" Dialog Box

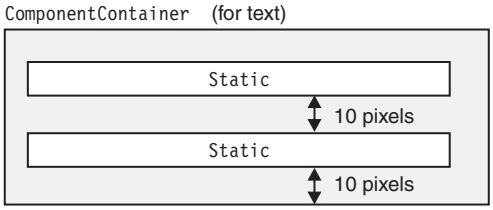
The first sample XML resource file creates an "About MyApp" dialog:

```
<?xml version="1.0" ?>
<Dialog name="aboutDialog" title="About MyApp"
   height="150" width="400" layout="vertical">

   <ComponentContainer layout="vertical" alignment="center">
      <Static name="credit"
       title="Brought to you by the Information Technology group"
       themeClass="csLblFnt csThmClr csAbtTxt" />

      <!-- Add a 10px space in between two Static components -->
      <Spacer />
      <Static name="company"
       title="Copyright 2003 Your company name here."
       themeClass="clsIbar" />
      <Spacer />
   </ComponentContainer>

   <!-- Add another ComponentContainer to have the button aligned in
        the center -->
   <ComponentContainer layout="horizontal" alignment="center">
      <!--If button text is less than 7-8 characters, add width="70" -->
      <Button name="ok" title="OK" />
   </ComponentContainer>
   <!-- Add 10px margin from bottom -->
   <Spacer />
</Dialog>
```

- The dialog box shown in the image below has a height of 150 pixels and a width of 400 pixels. The elements contained in this dialog are to be stacked vertically (`layout="vertical"`). If the width or height of a spacer is not specified, the default is 10 pixels in height and 10 pixels in width.



- The first `ComponentContainer` contains a credit statement (`Static`) and a company/copyright statement (`Static`).
- The two Static components are stacked vertically (`layout="vertical"`) and aligned in the center (`alignment="center"`).
- Some CSS classes are applied to the `Static` components. The theme classes are described in the Presenting Data chapter in the *Developer's Guide*.



- The second ComponentContainer contains a Button. In order to align the button in the center, it needs to be in its own ComponentContainer or it will align on the left with the two Static components.

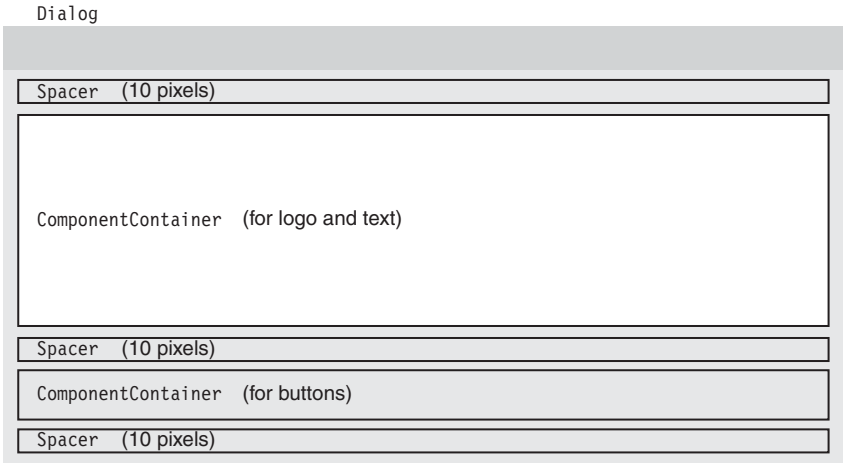
Example 2: A Confirmation Dialog Box

The second sample XML resource file creates a confirmation dialog:

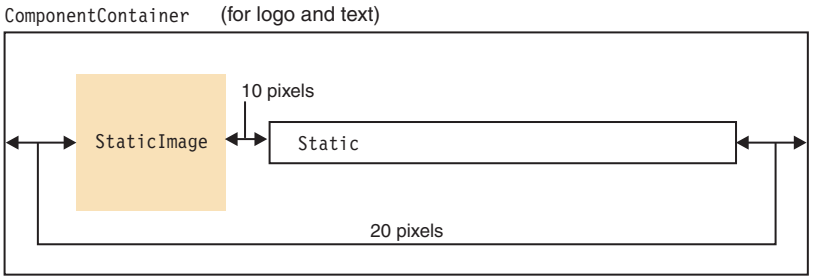
The XML code that produces the above dialog is as follows

```
<?xml version="1.0" ?>
<Dialog name="myDialog" title="Confirmation" modal="true"
   height="140" width="400" layout="vertical">

   <!-- Add 10px margin from top -->
   <Spacer />
   <!-- Need a horizontal layout in the main area in order to add 20px
        margin on each side -->
   <ComponentContainer layout="horizontal">
      <!-- Add 20px margin on left side -->
      <Spacer width="20" />

      <!-- CONTENT AREA-->
      <StaticImage imageURL="/SalesApp/images/logo.gif" />
      <Spacer width="10" />
      <Static name="credit"
         title="Do you want to apply the change?"
         themeClass="csLblFnt csThmClr csAbtTxt" />

      <!-- Add 20px margin on right side -->
      <Spacer width="20" />
   </ComponentContainer>

   <!-- Add 10px margin between content area and buttons -->
   <Spacer />

   <ComponentContainer name="buttonContainer" layout="horizontal"
    alignment="right">
      <!--If button text is less than 7-8 characters, add width="70" -->
      <Button name="ok" title="Yes" width="70" />

      <!-- Add 5px margin between buttons -->
      <Spacer width="5" />
      <Button name="cancel" title="Cancel" width="70" />
      <!-- Add 20px margin on right side, matching main content area -->
      <!-- Only put this in if there is equivalent or greater margin on the
           left already -->
      <Spacer width="20" />
   </ComponentContainer>

<!-- Add 10px margin from bottom -->
   <Spacer />
</Dialog>
```

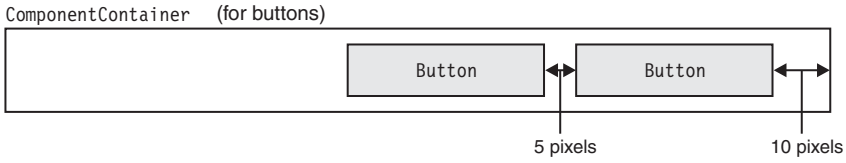
- The dialog box has a height of 150 pixels and a width of 400 pixels. The elements contained in this dialog are to be stacked vertically (`layout="vertical"`).



- A `ComponentContainer` is added to include a logo (`StaticImage`) and text (`Static`).
  - The `ComponentContainer's` layout is set to `horizontal` so components inside this container are stacked from left to right.
  - Some CSS classes are applied to the components. The theme classes are described in the Presenting Data chapter in the *Developer's Guide*.



- Another `ComponentContainer` is added to contain two buttons (`Button` components). The two buttons are right aligned (`alignement="right"`) with 10 pixels at the end to add some space before the border.



Element Attributes

This section lists the attributes common to all elements:

- "Common Attributes to All UI Elements" on page 429
- "Additional Attributes for CheckBox and RadioButton" on page 431
- "Additional Attributes for ControlbarItem, MenuItem, and ToolbarButton" on page 431

Common Attributes to All UI Elements

The following table lists the attributes common to all elements:

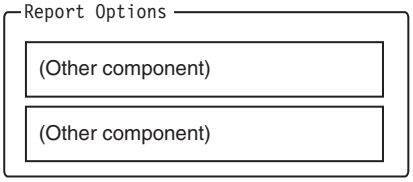
Attribute	Description
`name`	Specifies the name of the component. This is used to identify the component and provides the action name for the component. For example, if the component name is `myButton`, a ClickEvent on the component will invoke the method `actionMyButton`. **Important:** When naming your components, avoid reserved component names. Component names are the same as the constants in the *ModelConstants* interface in the com.alphablox.blox.unimodel package.
`title`	Specifies the title of the component. This is the displayed text. See "Use of the title Attribute" on page 430 for more information on the usage of the title attribute in each component.
`alignment`	Specifies the horizontal alignment of the component. Valid values are `right`, `left`, and `center`.
`batchEvents`	Specifies whether a corresponding event should be sent to the server immediately as the user takes an action that causes the componenet to change state. When this is set to `false`, as soon as the user takes an action that causes this component to change state, the client sends the event to the server. When this is set to `true`, events generated by this component are batched on the client until some other action causes the client to connect to the server (such as sending events from other components). This setting only effects events sent by components in dialogs.
`setBusyAfterEvent`	Specifies if the UI should be set to busy state each time the component generates an event. When set to `true`, UI of this component is set to busy when an event is generated. Valid values are `true` and `false`. This is useful in stopping user input during long or sensitive operations. This setting controls the behavior of the UI immediately after an event is generated.

`clickable`	Specifies if the component should generate mouse click events. Valid values are `true` and `false`.
`disabled`	Specifies whether the component is to be disabled. Valid values are `true` and `false`.
`height`	Specifies the height of the component in pixels.
`layout`	Specifies the layout to attach to the container. Valid values are `vertical`, `horizontal`, and `grid(`*numOfColumns*`)`. This attribute only applies to ComponentContainer and components deriving from it. For example, to create a layout of a four-column grid: layout=″grid(4)″
`setRightClickMenu`	Sets the right-click menu for this component. This menu will be displayed when users right-click the component.
`style`	Specifies the style to be attached to the component. The style can be expressed as a CSS-like style string. It can also be the name of a Style object.
`tabStop`	Sets the component as a tab stop. Valid values are `true` and `false.` The default is `true`. Note, however, that tab stop does not work with radio buttons. This is a browser behavior.
`themeClass`	Specifies the name of the theme class or classes that should be used for the component.
`tooltip`	Specifies the tool tip (popped-up text when users mouse over the component) to be attached to the component.
`valignment`	Specifies the vertical alignment of the component. Valid values are `top`, `bottom`, and `center`.
`visible`	Specifies the visibility of the component. Valid values are `true` and `false`.
`width`	Specifies the width of the component in pixels.

Use of the title Attribute

The following table describes the use of the `title` attribute in each element:

Element	**Use of the title Attribute**
`Button`	The button label.
`CheckBox`	The text displayed after the checkbox.
`ComponentContainer`	The title for top-level elements. Otherwise, it is ignored.
`Edit`	Ignored.
`GroupBox`	The title of the GroupBox. For example, with `title="Report Options"`, the display is as follows:



`Image StaticImage`	Ignored.
`ListBox DropDownList`	Ignored.
`Menu MenuItem`	The menu label.
`Menubar Toolbar`	Used in menus to refer to Toolbars. Otherwise, it is ignored.
`Spacer`	Ignored.
`Static`	The text to display.

Additional Attributes for CheckBox and RadioButton

Besides the common attributes described in "Common Attributes to All UI Elements" on page 429, the `CheckBox` and `RadioButton` have the following attributes:

Attribute	Description
`checked`	Specifies if the checkbox or radio button should be checked (selected). Valid values are `true` and `false`.

Additional Attributes for ControlbarItem, MenuItem, and ToolbarButton

Besides the common attributes described in "Common Attributes to All UI Elements" on page 429, the `ControlbarItem`, `MenuItem`, and `ToolbarButton` elements have the following attributes:

Attribute	Description
`checked`	Specifies if the MenuItem should be checked (selected). Valid values are `true` and `false`.
`checkBox`	Specifies if the MenuItem should act like a checkbox, with a check mark next to it when it is selected. Valid values are `true` and `false`.
`imageURL`	Specifies the image URL: • For absolute URLs, the string should begin with "`http://`". • For relative URLs, starting the string with a slash (`/`) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. Starting the string without a slash indicates that the URL is relative to the current theme directory. Typically, this directory is located at: `<alphablox>/repository/theme/<themeName>/i/`.
`separator`	Specifies that the item is a toolbar or menu separator. Valid values are `true` and `false`.

themeBasedImage	Specifies whether this image is located in the directory of the current theme. Valid values are true and false. When this attribute is set to true, the server looks for an image file that has the same name as the component, with a .gif extension.

Additional Attributes for Dialog

Besides the common attributes described in "Common Attributes to All UI Elements" on page 429, Dialog has the following attributes:

Attribute	Description
defaultButton	Specifies the default button that is clicked when users press the Enter key. All dialogs are created with the default button set to the OK button. Once users click another button, that button becomes the default.
modal	Specifies that the dialog is a model dialog. A model dialog lives in its own separate movable window and stops the rest of the UI from accepting input until it is dismissed.
resizable	Specifies that the user can resize the dialog window. Valid values are true and false. The default is false.

Additional Attributes for Image and StaticImage

Besides the common attributes described in "Common Attributes to All UI Elements" on page 429, the Image and StaticImage elements have the following attributes:

Attribute	Description
themeBasedImage	Specifies whether this image is located in the directory of the current theme. Valid values are true and false. When this attribute is set to true, the server looks for an image file that has the same name as the component, with a .gif extension in the theme's image directory. Typically, this directory is located at: <alphablox>/repository/theme/<themeName>/i
imageURL	Specifies the image URL as follows: • For absolute URLs, the string should begin with "http://". • For relative URLs, – Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL. – Starting the string without a slash indicates that the URL is relative to the current theme directory if themeBasedImage is set to true. If themeBasedImage is set to false, starting the string without a slash indicates that the URL is relative to the current application directory.

Additional Attributes for Static

Besides the common attributes described in "Common Attributes to All UI Elements" on page 429, the Static element has the following attributes:

Attribute	Description
`wrapText`	Specifies whether the text specified in the `title` attribute of a Static component should be wrapped. Valid values are `true` and `false`. The default is `false`.

Special Attribute for Top-level Component Containers

The five top-level component containers—Dialog, Menu, Menubar, Toolbar, and ComponentContainer— have one special attribute to specify caching of the resources:

Attribute	Description
`cache`	Specifies whether resources will be cached after loading. The default is `true`, meaning any changes made to the resource files require restarting the server.

Attributes for Item

The `Item` Element has only one attribute:

Attribute	Description
`value`	Specifies the item to be added to the list.

Attributes for ClientLink

The `ClientLink` element has three attributes. These attributes allow you to specify the same arguments you can pass in a `window.open()` JavaScript method.

Attribute	Description
`link`	Specifies the client link (a URL) for the component. Specification of this attribute is required.
`target`	Specifies the target window for the URL to load. This is optional. If not specified, the new URL is loaded into the same window.
`features`	Specifies the window features in a comma-separated string if the `target` attribute is specified. For example: `features="scrollbars=yes,width=300,height=300"`

Examples for Top-Level Elements

This section provides an example XML resource file for each of the top-level elements. For a complete example that demonstrates how to write a controller to control components created using an XML resource file, see the dial chart example in the Blox Sampler (DHTML version) under the UI Extensibility section.

ComponentContainer Element

The following XML is the actual resource file used by the Chart Types and Configuration dialog. This dialog has three tabs, one of which is Chart Types. When users click this tab, the following resource file is called:

```
<?xml version="1.0" ?>
<ComponentContainer name="chartTypesTab"
   title="Chart Types"
   layout="vertical"
   alignment="left">
   <Spacer />
   <ComponentContainer layout="horizontal">
      <Spacer width="20" />
      <ComponentContainer layout="vertical">
         <ComponentContainer layout="horizontal" alignment="left">
            <Static title="Chart Type" alignment="left" />
            <Spacer width="10" />
            <DropDownList name="chartTypesSelector" />
         </ComponentContainer>
         <Spacer height="5" />
         <Image name="chartTypeImage" />
      </ComponentContainer>
      <Spacer width="20" />
   </ComponentContainer>
</ComponentContainer>
```

The following code is the part of the XML in the Chart Types and Configuration dialog resource file that loads the above resource file:

```
<?xml version="1.0" ?>
<Dialog name="chartTypesDialog" title="Chart Types and Configuration"
   modal="false" height="420" width="450" layout="vertical">
   <TabbedContainer name="ChartTypesTabContainer"
      themeClass="csChrtCntnr" height="360" width="440"
      layout="horizontal" alignment="left">
   </TabbedContainer>
   <!--other components omitted -->
</Dialog>
```

Menu Element

The following XML creates a menu called `myFormatMenu`, with two menu items. This menu could then be, for example, attached to a component as a right-click menu.

```
<?xml version="1.0" ?>
<Menu name="myFormatMenu" title="Format" valignment="top">
   <MenuItem name="layout1" title="Special Layout 1"
      valignment="top" />
   <MenuItem name="layout2" title="Special Layout 2"
      valignment="top" />
</Menu>
```

Menubar Element

The following XML creates a View menu under a custom menu bar called `myMenubar`. The View menu has four menu options: Grid, Chart, Page Filter, and Data Layout. Following the View menu is a StaticImage that does not send any ClickEvent when clicked.

```
<?xml version="1.0" ?>
<Menubar name="myMenubar" layout="horizontal"
   themeClass="csCmpBg csThmClr csCmpBrdr csMnbr" >
   <Menu name="view" layout="vertical" title="View"/>
      <MenuItem name="viewGrid" title="Grid" checkBox="true"
          themeBasedImage="true" setBusyAfterEvent="true" />
```

```
      <MenuItem name="viewChart" title="Chart" checkBox="true"
          themeBasedImage="true" setBusyAfterEvent="true" />
      <MenuItem name="viewPageFilter" title="Page Filter"
          checkBox="true" themeBasedImage="true"
          setBusyAfterEvent="true" />
      <MenuItem name="viewDataLayout" title="Data Layout"
          checkBox="true" themeBasedImage="true"
          setBusyAfterEvent="true" />
   </Menu>
   <StaticImage name="logo" imageURL="smallLogo.gif"
      tooltip="Copyright (C) 2004 My Company"
      valignment="top" themeClass="clsStaticImage"
      themeBasedImage="true" />
</Menubar>
```

Dialog Element

See “Examples of Resource XML Files” on page 426 for a detailed layout discussion.

Toolbar Element

This sample XML resource file creates a Toolbar with the Excel, PDF, and Help buttons, with a separator between the PDF button and the Help button.

```
<Toolbar name="myToolbar" title="Exporting" layout="horizontal">

   <ToolbarButton name="fileExportToExcel" title="Excel"
    tooltip="Export to Excel"
    Bookmark" themeBasedImage="true" />

   <ToolbarButton name="fileExportToPDF" title="PDF"
    tooltip="Export to PDF"
    themeBasedImage="true" />

   <ToolbarButton separator="true" />

   <ToolbarButton name="helpHelp" title="Help" tooltip="Help"
    themeBasedImage="true" />

</Toolbar>
```

In this case, since `fileExportToExcel`, `fileExportToPDf`, and `helpHelp` are built-in ToolbarButtons (see “Custom Toolbar Tags” on page 400 for toolbar constants and values), we do not need to write our own controller to control these components. If you create your own ToolbarButton, a controller to handle the updates to and events coming from these components is needed. The images used for the buttons are theme-based, meaning they exist in the `<alphablox_dir>/repository/theme/i` directory, with different image files available for active, inactive, and disabled modes. You can also specify the URL to the image using the `imageURL` attribute. For more information on `themeBasedImage` and `imageURL`, see “The <bloxui:toolbarButton> Tag” on page 402.

Chapter 26. Client-Side API

This chapter contains reference material for client-side APIs available in the DHTML client. For information on how to use this reference, see Chapter 1, "Using This Reference," on page 1.

Client-Side API Overview

The Client API for the DHTML client has a relatively simple set of JavaScript objects, methods and events as compared to the other clients in DB2 Alphablox. The primary intent of the DHTML client is to enable easy access to server-side application logic and APIs. A summary of the four main components in the Client API are described next. For in-depth discussions of the client-side API and examples, see the *Developer's Guide* and accompanying examples in Blox Sampler.

Blox Methods

Each Blox has an associated JavaScript Blox object in the frame. When you create a PresentBlox with an `id` of "salesPresent", a JavaScript object of the same name is available on the page. A handful of JavaScript methods—such as `call()`, `getBloxAPI()`, `getName()`, `isBusy()`, `setBusy()`, `setDataBusy()`— allow you to call a JSP page on the server, update Blox properties on the server, set the busy state of a Blox and more.

These methods are common to user interface Blox (PresentBlox, GridBlox, ChartBlox, PageBlox, and ToolbarBlox) and are described in "Client-Side Blox Methods" on page 439.

BloxAPI

Each frame has one BloxAPI object that controls all incoming and outgoing traffic between the server and the client. This object is available to your JavaScript code through the global bloxAPI object. It provides methods to poll the server, send an event to the server, call a JSP page or invoke a method on a server-side bean, or add an event listener. The methods available to the bloxAPI object are described in "BloxAPI Methods" on page 439. For a complete example, see "Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method" on page 496.

Events

To intercept a user action such as loading a bookmark or swapping axes, you should use the associated server-side event filters (see the com.alphablox.blox.filter package in the Javadoc documentation). Alternatively, you can intercept the click event on the DHTML client before the event is sent to the server. For example, if you need to send a click event to the server, you can use the client API's (the `bloxAPI` object) `sendEvent` method:

```
bloxAPI.sendEvent( new ClientEvent( bloxname, uid, name) );
```

Sometimes it may be desirable or even necessary to intercept the events on the client-side before any server intervention. For example, if you need to cancel out a drillthrough event, using the server-side event filter's `cancelEvent()` method will only cancel the data operation, but not the pop-up window. Intercepting the event on the client-side allows you to cancel out the entire operation before it gets to the server.

Reference information on these client-side events are described in "Client-Side Events Reference" on page 450.

Custom Events

You can also create custom events that can be sent from the DHTML client to your components based on the Blox UI model on the server. This involves creating a class that extends the `ModelEvent` class in the `com.alphablox.blox.uimodel.core.event` package. See "Creating Custom Events" on page 454

Client Bean Registration using the Blox Header Tag

You can register a bean on the server for the server to remote the bean interface to the client in the form of a JavaScript object. This allows you to call any of the bean's methods from JavaScript just like any other JavaScript object. For example, if you have a bean called "myBean" on the server and it has the method:

```
String myMethod( String argument )
```

You can register the bean in the Blox header tag as follows:

```
<blox:header>
   <blox:clientBean name="myBean">
      <blox:method name="myMethod"/>
   </blox:clientBean>
</blox:header>
```

You can then call this registered method as follows:

```
function myFunction() {
   var result = myBean.myMethod( "somevalue" );
   alert(result);
}
```

For overloaded methods, append the method with an underscore, followed by the data type. For example, the following example registers a bean and its `setSelectableSlicerDimensions` method:

```
<blox:header>
   <blox:clientBean name="myDataBlox">
      <blox:method name="setSelectableSlicerDimensions"/>
   </blox:clientBean>
</blox:header>
```

The `setSelectableSlicerDimensions` method takes either an array of Dimension objects, or a String of dimensions. To call this method with a String:

```
<input type="button" value="Set Slicer Dimensions"
   onclick="myDataBloxAPI.setSelectableSlicerDimensions_String('Market,Measures');"
```

Note: This approach only works on primitive data types. For the list of data types supported, see "callBean()" on page 447.

See the *Developer's Guide* for in-depth discussions on the use of client beans.

DHTML Client API Cross References

This section provides cross reference tables for the following methods:

- "Client-Side Blox Methods" on page 439
- "BloxAPI Methods" on page 439
- "Client-Side Events and Event Methods" on page 440
- "Blox JavaScript Object Methods" on page 439

Client-Side Blox Methods

The following table lists the client-side JavaScript methods used to invoke server-side code from any user interface Blox on your page:

Methods
call()
getBloxAPI()
flushProperties()
getName()
isBusy()
setBusy()
setDataBusy()
updateProperties()

BloxAPI Methods

The following table lists the client-side JavaScript exposed via the `bloxAPI` Object:

Methods
"addBusyHandler()" on page 444
"addErrorHandler()" on page 445
"addEventListener()" on page 445
"addResponseListener()" on page 446
"call()" on page 446
"callBean()" on page 447
"getEnablePolling()" on page 448
"getPollingInterval()" on page 448
"poll()" on page 449
"sendEvent()" on page 449
"setEnablePolling()" on page 449
"setPollingInterval()" on page 450

Blox JavaScript Object Methods

The following table lists the client-side JavaScript exposed via the Blox JavaScript object. These methods are described in the Chapter 4, "Common Blox Tag Reference," on page 29.

JavaScript Methods
“call()” on page 441
“flushProperties()” on page 442
“getBloxAPI()” on page 442
“getName()” on page 442
“isBusy()” on page 443
“setBusy()” on page 443
“setDataBusy()” on page 443
“updateProperties()” on page 444

Client-Side Events and Event Methods

The following table lists all the client-side events available in the DHTML client:

Event
“ClickEvent” on page 451
“CaretPositionChangedEvent” on page 451
“ContentsChangedEvent” on page 451
“ClosedEvent” on page 451
“DoubleClickEvent” on page 451
“DragDropEvent” on page 451
“ExpandCollapseEvent” on page 451
“HScrollEvent” on page 451
“ResizeEvent” on page 451
“RightClickEvent” on page 451
“SelectedEvent” on page 451
“SelectionChangedEvent” on page 451
“UnselectedEvent” on page 451
“VscrollEvent” on page 451

The following table lists the methods common to all the client-side events:

Method
“getBloxName()” on page 452
“getDestinationName()” on page 452
“getDestinationUID()” on page 452
“getEventClass()” on page 452
“isReplaceDuplicate()” on page 452
“isUrgent()” on page 453
“setAttribute()” on page 453
“setReplaceDuplicate()” on page 453
“setUrgent()” on page 453

JavaScript Methods Common to Multiple Blox

This section describes the methods common to multiple Blox that are not associated with a specific property. To see if a method is valid for a particular Blox, see the methods section for that Blox.

call()

Calls a URL to execute on the server and returns the results of the HTTP request as a String. This method is useful to execute server-side code from the client. The `call()` method executes the server-side code without refreshing the page.

Data Sources

All

Syntax

JavaScript Method

```
call(callURL); // returns String
```

where:

Argument	Description
`callURL`	A String containing a URL of a file (typically a JSP file) to be run on the server.

Usage

Use the `call()` method to execute, from a client-side method, server-side code. You might use this method to set properties on the server or execute other server-side logic. This code is executed without refreshing the page (if your application is rendered in `html` mode, the `call()` method does cause a page refresh).

The `call()` method automatically flushes any pending transactions on the Blox, ensuring that any properties that have been set by users have propagated down to the server.

The `callURL` string can reference a JSP file that does not actually send anything to the client, but just performs various server actions. The URL can be absolute or relative:

- For absolute URLs, the string should begin with "`http://`".
- For relative URLs:
  - Starting the string with a slash (/) indicates that the URL is relative to the server root. Note that the application context needs to be included in the URL.
  - Starting the string without a slash indicates that the URL is relative to the current document.

Note that for absolute URLs, if the render mode is Java, you must call the same server that delivered the applet. This is due to the Java applet security policy.

Note: The default encoding of the response text from the `call()` method is UTF-8 if not otherwise specified. If you need a different encoding, specify your encoding in your JSP page directive, for example:

```
<%@ page contentTyp"text/html; charset=SHIFT_JIS" %>
```

Examples

```
myPresent.call("http://myserver/myapp/RunSomeCode.jsp"); //absolute URL
myPresent.call("/myapp/RunSomeCode.jsp");   //relative to server root
```

See Also

"BloxAPI Methods" on page 439, "setDataBusy()" on page 443; "Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method" on page 496.

flushProperties()

Ensures that all properties set on the client (for example, through user actions in the user interface) are propagated ("flushed") to the server.

Data Sources

All

Syntax

JavaScript Method

```
flushProperties(); // no return value
```

Usage

This method flushes to DB2 Alphablox all pending property changes for all of the Blox, so you only need to call it from one Blox to flush all properties on all Blox.

See Also

"call()" on page 441, "updateProperties()" on page 444

getBloxAPI()

Returns the global framework object.

Data Sources

All

Syntax

JavaScript Method

```
BloxAPI getBloxAPI();
```

Usage

This BloxAPI object can also be obtained using the `bloxAPI` variable available in each frame.

See Also

Chapter 26, "Client-Side API," on page 437

getName()

Returns the name of the Blox.

Data Sources

All

Syntax

JavaScript Method

```
String getName();
```

isBusy()

Returns the current busy state for the Blox.

Data Sources

All

Syntax

JavaScript Method

```
boolean isBusy();
```

Usage

Returns `true` if the Blox is busy; `false` if not.

See Also

"setBusy()" on page 443, Chapter 26, "Client-Side API," on page 437

setBusy()

Temporarily controls a Blox's busy state on the client.

Data Sources

All

Syntax

JavaScript Method

```
void setBusy(boolean busy);
```

where:

Argument	Default	Description
`busy`	none	When `true`, the Blox will indicate its busy state by disabling interactivity in its user interface and animating the logo in the menu bar.

See Also

"isBusy()" on page 443, Chapter 26, "Client-Side API," on page 437

setDataBusy()

Set the busy state on the client. This method is useful when you execute some server-side code that performs data operations.

Availability

Render Modes All

Data Sources All

Syntax

JavaScript Method

```
setDataBusy(state); // no return value
```

where:

Argument	Default	Description
state	false	A boolean argument. A value of true sets the state of the client-side blox to busy, thus disallowing any client-side code to execute. A value of false indicates that the state is not busy so client-side code can freely execute.

Usage
Set the setDataBusy() method on a client-side DataBlox that will be updated by the any server-side code, for example, code that is executed via the call() method.

See Also
"call()" on page 441

updateProperties()

Sends a refresh message to DB2 Alphablox to update the current Blox properties. This causes the Blox to transmit its working state (client-side) properties to the server-side Blox peer, thus insuring consistency between the server-side peer and the client-side Blox.

Data Sources
All

Syntax
JavaScript Method

```
updateProperties();    // no return value
```

See Also
"call()" on page 441, "flushProperties()" on page 442

BloxAPI Reference

The following methods are global methods available on the BloxAPI object in the frame.

addBusyHandler()

Adds a busy handler for all Blox on the page.

Syntax
JavaScript Method

```
addBusyHandler(busyHandler);
```

where:

Argument	Description
busyHandler	The name of a JavaScript function.

Usage
Busy handlers are invoked whenever a Blox's busy state changes. The busy handler provided will be called with the Blox object. This allows you to supply your own custom handler. The busy handler is a JavaScript function of the form:

```
boolean busyHander( Blox blox );
```

Note: The default action will gray out the Blox when busy. Return `true` to prevent further processing of the state changes.

addErrorHandler()

Adds an error handler to the framework.

Syntax

JavaScript Method

```
addErrorHandler(eventListener);
```

where:

Argument	Description
`eventListener`	The name of a JavaScript function.

Usage

Error handlers are invoked when the framework receives a communication or Simple Object Access Protocol (SOAP) error response from the server. The error handler is a JavaScript function of the form:

```
boolean errorHandler( SoapResponse response )
```

where `SoapResponse` has the following method and attributes:

- `boolean SoapResponse.hasFault();`
- `String SoapResponse.faultReason;`
- `String SoapResponse.faultCode;`
- `String SoapResponse.faultSubcode;`

The error handler should return `true` if it has handled the error and should stop any further processing of the error. If the error handler returns `false`, the error is sent to any remaining error handlers in the list. In any case, after an error, the framework returns without updating Blox UI contents or busy states.

addEventListener()

Adds an event handler to the framework.

Syntax

JavaScript Method

```
addEventListener(eventListener);
```

where:

Argument	Description
`eventListener`	The name of a JavaScript function.

Usage

Event listeners are invoked for each event sent by either the DHTML client code or by a developer using the sendEvent() method. The event listener is a JavaScript function of the form:

```
boolean eventListener( [ClientEvent] event )
```

where `ClientEvent` is any client event as listed in "Client-Side Events Reference" on page 450. The listener should return `true` to stop all further processing of the event. If the event handler returns `false`, the client API continues to process the event by sending the event to any remaining listeners and then on to the server.

addResponseListener()

Adds a response listener for all Blox on the page.

Syntax

JavaScript Method

```
addResponseListener(responseListener);
```

where:

Argument	Description
`responseListener`	The name of a JavaScript function.

Usage

This method lets you add a JavaScript function which will be notified whenever a DHTML client-RPC returns a success response. This can be used to coordinate actions across frames or other post-event processing. The JavaScript function gets both the request and the response

Examples

The following example demonstrates how the request and response are captured:

```
<%@ taglib uri='bloxtld' prefix='blox'%>

<html>
<head>
<blox:header />
<script>
  function responseListener( request, response ) {
    var text = "REQUEST:\r\n\r\n" + request.getRequest() + "\r\n\r\n-----
\r\n\r\n";
    text += "RESPONSE: \r\n\r\n" + response.getResponse();
    responseOutput.value = text;
  }
  bloxAPI.addResponseListener( responseListener );
</script>
</head>
...
<body>
<blox:present id="present"
   ...>
</blox:present>
...
<textarea id=responseOutput rows=100 cols=200>
...
</body>
</html>
```

call()

Makes an HTTP request to the supplied URL and returns the results of the request as a String.

Syntax

JavaScript Method

```
call(url);
```

where:

Argument	Description
`url`	A String containing a URL of a file (typically a JSP file) to be run on the server.

Usage

This method also polls the server for changes immediately after the HTTP request and before the method returns the results.

callBean()

Invokes the specified method on the indicated server-side bean.

Syntax

JavaScript Method

```
callBean(beanName, methodName);
callBean(beanName, methodName, argumentArray, argumentTypeArray);
```

where:

Argument	Description
`beanName`	The name of bean. The bean must have been previously registered with the HttpSession on the server.
`methodName`	The name of a bean method.
`argumentArray`	An array of all arguments to be passed to the bean method
`argumentTypeArray`	Optional. An array of the data type of the argument(s). This helps match the arguments to the proper method on the client-side bean. See the table below for the data types supported.

Usage

The return value will be converted to an appropriate JavaScript data type. If the bean method throws a Java Exception, the return value will be a JavaScript Exception object. See the *Developer's Guide* for more on the JavaScript Exception object.

Supported method argument types are case-sensitive and are limited to the following primitive data type:

Java Data Type	Valid `argumentTypes` Value
String	`string` or `unspecified`
Integer	`integer` or `int`
Boolean	`boolean`
Long	`long`
Double	`double`
Float	`float`
Byte	`byte`
Array	`JavaScript array`

If an argument is not typed and the server-side bean does not have a method signature matching the argument listed, the server will look for methods that take Java objects other than primitive types. In this case, the server will try to create the required object using a String-based constructor for the object.

Examples

The following example calls the myMethod method on a bean called myBean on the server with two arguments: a string and an integer.

```
var results = bloxAPI.callBean('myBean', 'myMethod', new Array('arg1',
'2'), newArray('string', 'int');
```

If you have a server bean myBean with a single method as follows:

```
String beanMethod( MyObject object ) // myBean's method signature
```

If you call the above bean method from the client using the following statement without specifying the argument's data type:

```
bloxAPI.callBean( "myBean", "beanMethod", "foobar" );
```

The server will invoke the method as follows:

```
myBean.beanMethod( new MyObject("foobar"));
```

Notice the creation of a MyObject object uses a String-based constructor. If the MyObject object's String-based constructor can make sense of the supplied value from the client, the method will be invoked and the results returned to the client.

Important: It is recommended that you limit client-side code to calling only methods that take and return primitive data types rather than Java objects.

getEnablePolling()

Returns the polling enabled setting.

Syntax

JavaScript Method

```
getEnablePolling();
```

Usage

If true, the automatic polling mechanism is enabled and the framework underlying the DHTML client does a very slow poll of the server designed to pick up, in very rare occasions, any asynchronous changes to the Blox in the frame.

See Also

"setEnablePolling()" on page 449

getPollingInterval()

Returns the polling interval for non-busy polling in milliseconds.

Syntax

JavaScript Method

```
getPollingInterval();
```

Usage

This is the normal polling interval that checks the server for asynchronous updates. The polling mechanism uses a different interval when the server informs the client that it is busy.

See Also

"setPollingInterval()" on page 450

poll()

Immediately polls the server for changes to any of the Blox in the frame.

Syntax

JavaScript Method

```
poll();
```

Usage

The server will respond with any pending changes as well as the busy state for each of the Blox. Normally you do not need to explicitly poll the server since the framework handles this automatically. This is only needed if you modify the state of the server. In this case, polling may be needed in order to pick up those changes in a timely manner. Typically this means circumventing the DHTML client in some way such as using multiple frames.

sendEvent()

Immediately sends the named event to all registered event handlers and then ultimately to the server.

Syntax

JavaScript Method

```
sendEvent(ClientEvent event);
```

where:

Argument	Description
`ClientEvent`	One of the JavaScript event objects: `ClickEvent`, `ContentsChangedEvent`, `ClosedEvent`, `DoubleClickEvent`, `DragDropEvent`, `RightClickEvent`, `ScrollEvent`, `SelectedEvent`, `SelectionChangedEvent`, `UnselectedEvent`.
`event`	The name of the event.

Usage

If the client successfully sends the event to the server, the function returns `true`. In all other cases, the function returns `false`.

setEnablePolling()

Controls automatic server polling.

Syntax

JavaScript Method

```
setEnablePolling(enable);
```

where:

Argument	Description
`enable`	Specify `true` to enable the automatic polling mechanism; `false` to disable it.

Usage

When set to `false`, the framework underlying the client will not automatically poll the server. The automatic polling mechanism is a very slow poll of the server

designed to pick up, in very rare occasions, any asynchronous changes to the Blox in the frame. By default, polling is enabled.

See Also

"getEnablePolling()" on page 448

setPollingInterval()

Sets the polling interval for non-busy polling in milliseconds.

Syntax

JavaScript Method

```
setPollingInterval(intervalMS);
```

where:

Argument	Description
`intervalMS`	The non-busy polling interval in milliseconds.

Usage

This is the normal polling interval that checks the server for asynchronous updates. The polling mechanism uses a different interval when the server informs the client that it is busy.

See Also

"getPollingInterval()" on page 448

Client-Side Events Reference

The Blox UI model exposes a set of events as JavaScript objects. As a result, you can use JavaScript to create event objects, send the events to the server, or intercept these events. Each event has a class name that matches the name of the class on the server used to dispatch the event. The class name is always available in the `EVENT_CLASS` static attribute on each event. For example, for ClickEvent, the class name is `ClickEvent.EVENT_CLASS`. For SelectionChangedEvent, the class name is `SelectionChangedEvent.EVENT_CLASS`. All the events have a common set of methods that let you identify the destination Blox name or component UID, specify whether an event should be dispatched immediately, and more.

This section provides information on the following topics:

Client-Side Events and Syntax

The following table lists the client-side event objects available in the DHTML client. Note that the argument types are provided to help describe the arguments. In JavaScript, all arguments are variables.

Event	Syntax
ClickEvent	ClickEvent(String *bloxName*, int *uid*, String *name* [, Boolean *checked*]);
CaretPositionChangedEvent	CaretPositionChangedEvent(String *bloxName*, int *uid*, String *name*, int *caretPosition*, String *textSelection*);
ContentsChangedEvent	ContentsChangedEvent(String *bloxName*, int *uid*, String *name*, *newContents*);
ClosedEvent	ClosedEvent(String *bloxName*, int *uid*, String *name*);
DoubleClickEvent	DoubleClickEvent(String *bloxName*, int *uid*, String *name*);
DragDropEvent	DragDropEvent(String *bloxName*, int *uid*, *operation*, *droppedComponent* [, int *positionAfterComponentUID*]);
ExpandCollapseEvent	ExpandCollapseEvent(String *bloxName*, int *uid*, String *name*, boolean *expanded*);
HScrollEvent	HScrollEvent(String *bloxName*, int *uid*, String *name*, int *newHorizontalPosition*);
ResizeEvent	ResizeEvent(String *bloxName*, int *uid*, String *name*, int *newWidth*);
RightClickEvent	RightClickEvent(String *bloxName*, int *uid*, String *name*);
SelectedEvent	SelectedEvent(String *bloxName*, int *uid*, String *name*);
SelectionChangedEvent	SelectionChangedEvent(String *bloxName*, int *uid*, String *name*, Array *integerSelections*);
UnselectedEvent	UnselectedEvent(String *bloxName*, int *uid*, String *name*);
VScrollEvent	VScrollEvent(String *bloxName*, int *uid*, String *name*, int *newVerticalPosition*);

The arguments are described below:

Arguments

`bloxName`	The name of the Blox the event should be sent to.
`uid`	The unique id associated with the component on which the event will be called.
`name`	Optional. The component name which is the actual text-based name of the component generating the event. In the cases where `name` is not needed and yet it is not the last argument, it should be set to null.
`caretPosition`	Location of the text cursor.
`checked`	Optional. The current state of the checkbox on the client (for CheckBox components).
`droppedComponent`	The `uid` of the component that is being dropped.
`expanded`	`true` if expanded.
`integerSelections`	Either an array of integers containing a list of `uid` of the selections or an array of integers containing the indices for multiple selection lists.
`newContents`	The changed content. This is usually a string such as that entered in an Edit component (an text edit box).

`newHeight`	The new height in pixels.
`newHorizontalPosition`	A 0-based horizontal scroll index. The scroll unit is defined by the component using the event. For example, for the grid component, this would be the column number.
`newVerticalPosition`	A 0-based vertical scroll index. The scroll unit is defined by the component using the event. For example, for the grid component, this would be the row number.
`newWidth`	The new width in pixels.
`operation`	The operation performed. Currently, `move` is only valid operation.
`positionAfterComponentUID`	Optional. The UID of the component the target component should be dropped after.
`textSelection`	The current selected text that is marked with the mouse.

Common Event Methods

Each event exposes the following methods:

getBloxName()

Returns the destination Blox name.

Syntax:

```
String getBloxName();
```

getDestinationName()

Returns the destination component's name.

Syntax:

```
String getDestinationName();
```

Usage: You should use `getDestinationUID()` whenever possible as this method may return null for some components. Use this only in cases when `getDestinationUID()` does not work for you (for example, when you need to identify whether the destination component is your custom component).

getDestinationUID()

Returns the destination component's UID.

Syntax:

```
int getDestinationUID();
```

getEventClass()

Returns the server-side Java class name associated with the event.

Syntax:

```
String getEventClass();
```

isReplaceDuplicate()

Returns `true` if this event will replace duplicate events in the client-side event queue.

Syntax:

```
boolean isReplaceDuplicate();
```

isUrgent()

Returns `true` if this is an urgent event to be sent immediately to the server

Syntax:

```
boolean isUrgent();
```

setAttribute()

Sets the value of the attribute name in the event.

Syntax:

```
void setAttribute( String name, String value [, String type] );
```

Usage: The attribute value is passed along with the event to the server. The type of the attribute can be specified using the optional type argument. For supported data type, see "Support Data Type" on page 447.

setReplaceDuplicate()

Specifies whether an event of the same event class, destination UID and destination Blox name in the client-side event queue should be replaced.

Syntax:

```
void setReplaceDuplicate( boolean replaceDuplicate);
```

Usage: When `true`, an event of the same event class, destination UID and destination Blox name residing in the client-side event queue (that is, non-urgent events) will get replaced. The process removes the existing duplicate event from the queue and adds the replacement to the end of the queue.

setUrgent()

Specifies to sent urgent events to the server immediately.

Syntax:

```
void setUrgent( boolean isUrgent );
```

Usage: Urgent events are sent to the server immediately with no waiting. Non-urgent events are sent when convenient, such as during a poll or other server communications including the sending of an urgent event.

Generating an Event

To generate an event:

```
var myClickEvent = new ClickEvent (bloxName, uid [, componentName] };
```

Then to send an event:

```
bloxAPI.sendEvent( myClickEvent );
```

To intercept an event:

```
bloxAPI.addEventListener(eventHandler);
```

and the eventHandler JavaScript function can be something as shown in the following example:

```
<script>
function eventHandler(event) {
  alert( "At handler for event " + event.getEventClass() +
  " on component UID " + event. getDestinationUID() );
  return false;
}
</script>
```

Returning `false` from the handler will allow the event to be processed and sent to the server. Returning `true` will stop all further processing of the event.

The following JavaScript example demonstrates how to make sure that events occurring on a custom UI component (in this example, the component's name is ″Show″) get dispatched immediately:

```
function eventListener( event )
{
  if ( event.getDestinationName() == "Show" )
  {
    // Make this event not dispatch immediately
    event.setUrgent( false );

    // Set busy on the blox
    myShowContainer.setBusy( true );
    myGrid.setBusy( true );

    // After a bit of time, make sure the event is sent out
    setTimeout( "bloxAPI.flushEvents();", 0 );
  }
  return false;
}

bloxAPI.addEventListener( eventListener );
```

For more details and examples see the DHTML Client API chapter in the *Developer's Guide*.

Creating Custom Events

You can create you own custom events that work like the supplied events. Custom events should use the existing client-side event infrastructure if these events are to be sent from the client. The recipe for generating client-side custom events is as follows:

1. Define a JavaScript function with the name of the event.
2. Add additional event parameters to the constructor.
3. Set the class name of the event on the function by setting the `EVENT_CLASS` attribute.
4. Call the `_modelEventConstructor()` in the function.
5. Set any additional parameters on the event using the `setAttribute()` method.

```
function MyEvent( bloxName, uid, name, myvalue )
{
   // Begin constructor
   _modelEventConstructor( this, MyEvent.EVENT_CLASS, bloxName, uid, name );
   this.setAttribute( "MyEvent.myValue", myvalue, "int" );
   // End constructor
}
MyEvent.EVENT_CLASS = "my.package.MyEvent";
```

Assuming that the event my.package.MyEvent has been defined on the server:

```
package my.package;
import com.alphablox.blox.uimodel.core.event.ModelEvent;
Public class MyEvent extends ModelEvent
{
  ...
}
```

Appendix A. JSP Custom Tag Copy and Paste

This appendix contains versions of the custom tag libraries for each blox. You can use these versions to copy and paste into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need.

AdminBlox JSP Custom Tag

Below shows the entire AdminBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for the attributes in your JSP file or the page will not compile.

```
<blox:admin
       id=""
       bloxName=""
/>
```

BookmarksBlox JSP Custom Tag

Below is the entire BookmarksBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "BookmarksBlox Tag Attributes" on page 63.

```
<blox:bookmarks
       id=""
       bloxName=""
/>
```

ChartBlox JSP Custom Tag

Below shows the entire ChartBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "ChartBlox Tag Attributes" on page 76.

```
<blox:chart
       id=""
       absoluteWarning=""
       aggregateIdenticalInstances=""
       applyPropertiesAfterBookmark=""
       areaSeries=""
       autoAxesPlacement=""
       axisTitleStyle=""
       backgroundFill=""
       barSeries=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       chartAbsolute=""
       chartCurrentDimensions=""
       chartFill=""
       chartType=""
       columnLevel=""
       columnSelections=""
       comboLineDepth=""
       contributionParameters=""
       dataTextDisplay=""
       dataValueLocation=""
       depthRadius=""
       dwellLabelsEnabled=""
       filter=""
       footnote=""
       footnoteStyle=""
       formatProperties=""
       gridLineColor=""
       gridLinesVisible=""
       groupSmallValues=""
       height=""
       helpTargetFrame=""
       histogramOptions=""
       labelStyle=""
       legend=""
       legendPosition=""
       lineSeries=""
       lineWidth=""
       logScaleBubbles=""
       markerShape=""
       markerSizeDefault=""
       maxChartItems=""
       maximumUndoSteps=""
       menubarVisible=""
       mustIncludeZero=""
       noDataMessage=""
       o1AxisTitle=""
       pieFeelerTextDisplay=""
       quadrantLineCountX=""
```

```
        quadrantLineCountY=""
        quadrantLineDisplay=""
        removeAction=""
        render=""
        rightClickMenuEnabled=""
        riserWidth=""
        rowHeaderColumn=""
        rowLevel=""
        rowSelections=""
        rowsOnXAxis=""
        seriesColorList=""
        showSeriesBorder=""
        smallValuePercentage=""
        title=""
        titleStyle=""
        toolbarVisible=""
        totalsFilter=""
        trendLines=""
        useSeriesShapes=""
        visible=""
        width=""
        x1AxisTitle=""
        x1FormatMask=""
        x1LogScale=""
        x1ScaleMax=""
        x1ScaleMaxAuto=""
        x1ScaleMin=""
        x1ScaleMinAuto=""
        XAxis=""
        XAxisTextRotation=""
        y1Axis=""
        y1AxisTitle=""
        y1FormatMask=""
        y1LogScale=""
        y1ScaleMax=""
        y1ScaleMaxAuto=""
        y1ScaleMin=""
        y1ScaleMinAuto=""
        y2Axis=""
        y2AxisTitle=""
        y2FormatMask=""
        y2LogScale=""
        y2ScaleMax=""
        y2ScaleMaxAuto=""
        y2ScaleMin=""
        y2ScaleMinAuto=""
>
</blox:chart>
```

Nested Tags Inside <blox:chart>

```
<blox:chart ...>
   <blox:footnoteStyle
      font=""
      foreground="" />
   <blox:labelStyle
      font=""
      foreground="" />
   <blox:seriesFill
      index=""
      value="" />
   <blox:contribution
      baseSeries=""
      contributionSeries=""
```

```
      rootGroup=""
      sortType=""
      thresholdType=""
      thresholdValue="" />
   <blox:titleStyle
      font=""
      foreground=""/>

   <blox:dial
      borderColor=""
      borderType=""
      color=""
      formatMask=""
      radius=""
      showLabels=""
      startAngle=""
      stopAngle=""
      ticPosition="">
      <blox:needle
         color=""
         endType=""
         endWidth=""
         needleWidth=""
         scope=""
         tooltip=""
         value="" />
      <blox:scale
         maximum=""
         minimum=""
         scope=""
         stepSize="" />
      <blox:sector
         color=""
         innerRadius=""
         outerRadius=""
         scope=""
         startValue=""
         stopValue=""
         tooltip="" />
   </blox:dial>

</blox:chart>
```

CommentsBlox JSP Custom Tag

This section shows the entire CommentsBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "CommentsBlox Tag Attributes" on page 147.

```
<blox:comments
       id=""
       bloxName=""
       bloxRef=""
       commentsOnBaseMember=""
       collectionName=""
       dataSourceName=""
       userName=""
       password="" >
```

```
       <blox:sortComments
          field=""
          order="" />
</blox:comments>
```

ContainerBlox JSP Custom Tag

This section shows the entire ContainerBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

```
<blox:container
       id=""
       bloxName=""
       enablePoppedOut=""
       height=""
       poppedOut=""
       poppedOutHeight=""
       poppedOutTitle=""
       PoppedOutWidth=""
       render=""
       visible=""
       width=""
>
</blox:container>
```

DataBlox JSP Custom Tag

This section shows the entire DataBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "DataBlox Tag Attributes" on page 161.

```
<blox:data
       id=""
       bloxRef=""
       aliasTable=""
       applyPropertiesAfterBookmark=""
       autoConnect=""
       autoDisconnect=""
       bloxName=""
       bookmarkFilter=""
       calculatedMembers=""
       catalog=""
       columnSort=""
       connectOnStartup=""
       credential=""
       dataSourceName=""
       dimensionRoot=""
       drillDownOption=""
       drillKeepSelectedMember=""
       drillRemoveUnselectedMembers=""
       enableKeepRemove=""
       enableShowHide=""
       hiddenMembers=""
       hiddenTuples=""
       internalSortEnabled=""
       leafDrillDownAvailable=""
       memberNameRemovePrefix=""
       memberNameRemoveSuffix=""
       mergedDimensions=""
```

```
       mergedHeaders=""
       onErrorClearResultSet=""
       parentFirst=""
       password=""
       performInAllGroups=""
       provider=""
       query=""
       retainSlicerMemberSet=""
       rowSort=""
       schema=""
       selectableSlicerDimensions=""
       showSuppressDataDialog=""
       suppressDuplicates=""
       suppressMissingColumns=""
       suppressMissingRows=""
       suppressNoAccess=""
       suppressZeros=""
       textualQueryEnabled=""
       useAASUserAuthorization=""
       useAliases=""
       useOlapDrillOptimization=""
       userName=""
</blox:data>
```

DataLayoutBlox JSP Custom Tag

This section shows the entire DataLayoutBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "DataLayoutBlox Tag Attributes" on page 212.

```
<blox:dataLayout
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       height=""
       helpTargetFrame=""
       hiddenDimensionsOnOtherAxis=""
       interfaceType=""
       maximumUndoSteps=""
       noDataMessage=""
       render=""
       visible=""
       width=""  >
</blox:dataLayout>
```

GridBlox JSP Custom Tag

This section shows the entire GridBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "GridBlox Tag Attributes" on page 222.

```
<blox:grid
       id=""
       applyPropertiesAfterBookmark=""
```

```
        autosizeEnabled=""
        bandingEnabled=""
        bloxEnabled=""
        bloxName=""
        bookmarkFilter=""
        columnHeadersWrapped=""
        columnWidths=""
        commentsEnabled=""
        defaultCellFormat=""
        drillThroughEnabled=""
        drillThroughWindow=""
        editableCellStyle=""
        editedCellStyle=""
        enablePoppedOut=""
        expandCollapseMode=""
        gridLinesVisible=""
        headingIconsVisible=""
        headingsEnabled=""
        height=""
        helpTargetFrame=""
        informationWindowName=""
        maximumUndoSteps=""
        menubarVisible=""
        missingValueString=""
        noAccessValueString=""
        noDataMessage=""
        poppedOut=""
        poppedOutHeight=""
        poppedOutTitle=""
        PoppedOutWidth=""
        relationalRowNumbersOn=""
        removeAction=""
        render=""
        rightClickMenuEnabled=""
        rowHeadersWrapped=""
        rowHeadingWidths=""
        rowHeight=""
        rowIndentation=""
        showColumnDataGeneration=""
        showColumnHeaderGeneration=""
        showRowDataGeneration=""
        showRowHeaderGeneration=""
        toolbarVisible=""
        visible=""
        width=""
        writebackEnabled="" >
</blox:grid>
```

Nested Tags Inside <blox:grid>

```
<blox:grid ...>
   <blox:cellAlert
      index=""
      apply=""
      background=""
      condition=""
      description=""
      enabled=""
      font=""
      foreground=""
      format=""
      group=""
      link=""
      image_align=""
```

```
   image=""
   scope=""
   value=""
   value2="" />]
<blox:cellEditor
   index=""
   scope="" />
<blox:cellFormat
   index=""
   background=""
   font=""
   foreground=""
   format=""
   group=""
   scope="" />
<blox:cellLink
   index=""
   description=""
   link=""
   scope=""
   image_align=""
   image="" />
<blox:drillThroughWindow
   height=""
   locationbarVisible=""
   menubarVisible=""
   name=""
   resizable=""
   scrollbarVisible=""
   statusbarVisible=""
   toolbarVisible=""
   url=""
   width="" />
<blox:editableCellStyle
   background=""
   font=""
   foreground="" />
<blox:editedCellStyle
   background=""
   font=""
   foreground="" />
<blox:formatMask
   index=""
   mask="" />
<blox:formatName
   index=""
   name="" />
```

MemberFilterBlox JSP Custom Tag

This section shows the entire MemberFilterBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "MemberFilterBlox Tag Attributes" on page 264.

```
<blox:memberFilter
       id=""
       applyButtonEnabled=""
       bloxEnabled=""
```

```
        bloxName=""
        dimensionSelectionEnabled=""
        height=""
        selectableDimensons=""
        selectedDimension=""
        visible=""
        width="" >
</blox:memberFilter>
```

PageBlox JSP Custom Tag

This section shows the entire PageBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "PageBlox Tag Attributes" on page 271.

```
<blox:page
        id=""
        applyPropertiesAfterBookmark=""
        bloxEnabled=""
        bloxName=""
        bookmarkFilter=""
        fixedChoiceLists=""
        height=""
        helpTargetFrame=""
        labelPlacement=""
        maximumUndoSteps=""
        menubarVisible=""
        moreChoicesEnabled=""
        moreChoicesEnabledDefault=""
        noDataMessage=""
        render=""
        visible=""
        width="" >
</blox:page>
```

PresentBlox JSP Custom Tag

This section shows the entire PresentBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "PresentBlox Tag Attributes" on page 278.

```
<blox:present
        id=""
        applyPropertiesAfterBookmark=""
        bloxEnabled=""
        bloxName=""
        chartAvailable=""
        chartFirst=""
        dataLayoutAvailable=""
        dividerLocation=""
        enablePoppedOut=""
        gridAvailable=""
        height=""
        helpTargetFrame=""
        maximumUndoSteps=""
        menubarVisible=""
```

```
        noDataMessage=""
        pageAvailable=""
        poppedOut=""
        poppedOutHeight=""
        poppedOutTitle=""
        PoppedOutWidth=""
        removeAction=""
        render=""
        splitPane=""
        splitPaneOrientation=""
        toolbarVisible=""
        visible=""
        width=""  >
</blox:present>
```

RepositoryBlox JSP Custom Tag

This section shows the entire RepositoryBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "RepositoryBlox Tag Attributes" on page 286.

```
<blox:repository
        id=""
        bloxName=""
        render="">
</blox:repository>
```

ResultSetBlox JSP Custom Tag

This section shows the entire ResultSetBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "ResultSetBlox Tag Attributes" on page 289.

```
<blox:resultSet
        id=""
        bloxName=""
        dataBlox=""
        resultSetHandler=""
/>
```

StoredProceduresBlox JSP Custom Tag

This section shows the entire StoredProceduresBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "StoredProceduresBlox Tag Attributes" on page 297.

```
<blox:storedProcedures
        id=""
        bloxName=""
/>
```

ToolbarBlox JSP Custom Tag

This section shows the entire ToolbarBlox custom tag library. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

For descriptions of each of the attributes, including the syntax for their values, see "ToolbarBlox Tag Attributes" on page 301.

```
<blox:toolbar
       id=""
       applyPropertiesAfterBookmark=""
       bloxEnabled=""
       bloxName=""
       bookmarkFilter=""
       helpTargetFrame=""
       removeAction=""
       removeButton=""
       rolloverEnabled=""
       textVisible=""
       toolTipsVisible=""
       visible=""
/>
```

Miscellaneous Tags in blox.tld

This section shows the remaining custom tag libraries. You can copy and paste this into your JSP files and remove the attributes you do not need and fill in the appropriate values for the ones you do need. You must fill in values for all the attributes in your JSP file or the page will not compile.

Display Tag

```
<blox:display
    blox=""
    bloxRef=""
    render=""
    width=""
    height="" />
```

Header Tag

```
<blox:header
    contextPath=""
    pageURL=""
    theme="" >
    <blox:clientBean name="" bean="">
        <blox:method name="" />
    </blox:clientBean>
</blox:header>
```

Blox Context Tag

```
<blox:bloxContext />
```

pdfReport and pdfDialogInput Tags

```
<blox:pdfReport
    header=""
    headerHeight=""
    footer=""
    footerHeight=""
    margin=""
```

```
    pageBreak=""
    size=""
    theme=""
    themeListEnabled="" >
    <blox:pdfDialogInput
        index=""
        displayName=""
        defaultValue=""
    />
</blox:pdfReport>
```

Debug Tag

```
<blox:debug />
```

For examples of using this tag, see "The Blox Debug Tag" in the *Developer's Guide*.

Logo Tag

```
<blox:logo />
```

Session Tag

```
<blox:session
    key="" />
```

Resource Bundle Related Tags

```
<blox:resourceBundle
    id=""
    bundle=""
    locale=""
    path="" />

<blox:message key=""/>

<blox:message key="">
    <blox:messageArg value="" />
</blox:message>
```

Blox Form-related Custom Tags

This section lists the tag attributes for:

CheckBoxFormBlox Tag

```
<bloxform:checkBox
   id=""
   bloxName=""
   checked=""
   checkedValue=""
   formElementName=""
   themeClass=""
   title=""
   uncheckedValue=""
   visible=""
/>
```

CubeSelectFormBlox

```
<bloxform:cubeSelect
   id="cubes"
   id=""
   bloxName=""
   dataBlox=""
   dataBloxRef=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   selectedCube=""
   selectedCubeName=""
   size=""
   themeClass=""
   visible=""
/>
```

DataSourceSelectFormBlox

```
<bloxform:dataSourceSelect
   id=""
   bloxName=""
   adapter=""
   adminBloxRef=""
   formElementName=""
   minimumWidth=""
   nullDataSourceLabel=""
   selectedDataSourceName=""
   themeClass=""
   type=""
   visible=""
/>
```

DimensionSelectFormBlox

```
<bloxform:dimensionSelect
   id=""
   bloxName=""
   cube=""
   cubeName=""
   dataBlox=""
   dataBloxRef=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   selectedDimension=""
   selectedDimensionName=""
   size=""
   themeClass=""
   visible=""
/>
```

EditFormBlox

```
<bloxform:edit
   id=""
   bloxName=""
   charactersPerLine=""
   focus=""
   formElementName=""
   lines=""
   maskInput=""
   maxCharacters=""
   themeClass=""
   visible=""
/>
```

MemberSelectFormBlox

```
<bloxform:memberSelect
   id=""
   bloxName=""
   dataBlox=""
   dataBloxRef=""
   dimension=""
   dimensionName=""
   filterGeneration=""
   filterOperator=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   rootMemberName=""
   rootMemberNames=""
   rootMembers=""
   selectedMember=""
   selectedMemberName=""
   size=""
   themeClass=""
   visible=""
/>
```

RadioButtonFormBlox

The <bloxform:radioButton> tag can have multiple <bloxform:button> tags.

```
<bloxform:radioButton
   id=""
   bloxName=""
   align=""
   borderEnabled=""
   formElementName=""
   themeClass=""
   visible="">

   <bloxform:button
      label=""
      object=""
      selected=""
      value=""
   />

</bloxform:radioButton>
```

SelectFormBlox

The <bloxform:select> tag can have multiple <bloxform:option> tags.

```
<bloxform:select
   id=""
   bloxName=""
```

```
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   size=""
   themeClass=""
   visible=""
>
   <bloxform:option
      label=""
      object=""
      selected=""
      value=""
   />

</bloxform:select>
```

TimePeriodSelectFormBlox

```
<bloxform:timePeriodSelect
   id=""
   bloxName=""
   defaultSeriesVisible=""
   formElementName=""
   minimumWidth=""
   selectedSeries=""
   selectedSeriesString=""
   themeClass=""
   timeSchemaBloxRef=""
   visible=""
>
   <bloxform:timeSeries
      expression=""
      name=""
   />

</bloxform:timePeriodSelect>
```

TimeUnitSelectFormBlox

```
<bloxform:timeUnitSelect
   id=""
   bloxName=""
   formElementName=""
   minimumWidth=""
   multipleSelect=""
   selectedTimeUnit=""
   size=""
   themeClass=""
   timeSchemaBloxRef=""
   visible=""
/>
```

TreeFormBlox

The <bloxform:tree> tag has two nested tags for folders and items. There can be multiple <bloxform:folder> tags and <bloxform:item> tags in a <bloxform:tree> tag.

```
<bloxform:tree
   id=""
   bloxName=""
   draggingEnabled=""
   itemPositoningEnabled=""
   rootVisible=""
   textWrapped=""
   themeClass=""
   visible=""
```

```
>
   <bloxform:folder> <%--root folder--%>

      <bloxform:folder
         label=""
         draggable=""
         expanded=""
         href=""
         imageURL=""
         label=""
         name=""
         object=""
         target=""
         themeBasedImage=""
         tooltip=""
      >
         <bloxform:item
            draggable=""
            href=""
            imageURL=""
            label=""
            name=""
            object=""
            target=""
            themeBasedImage=""
            tooltip="" />

      </bloxform:folder>

   </bloxform:folder>
</bloxform:tree>
```

The <bloxform:getChangedProperty> Tag

```
<bloxform:getChangedProperty
   debugEnabled=""
   formBlox=""
   formBloxRef=""
   formProperty=""
   property=""
/>
```

The <bloxform:setChangedProperty> Tag

```
<bloxform:setChangedProperty
   callAfterChange=""
   debugEnabled=""
   formProperty=""
   target=""
   targetRef=""
   targetProperty=""
/>
```

Blox Logic Custom Tags

The Blox Logic Tag Library has tags for the following Blox:

MDBQueryBlox

Tags for MDBQueryBlox has a nesting structure. The nesting structure may vary depending on application needs. For details, see "MDBQueryBlox Tags" on page 346. The following code shows the general structure:

```
<bloxlogic:mdbQuery
   id=""
   dataBloxRef=""
   cubeName="" >
   <bloxlogic:axis
      mutable=""
      queryFragment=""
      type="" >
      <bloxlogic:tupleList>
         <bloxlogic:dimension>
            [specify the dimension name here]
         </bloxlogic:dimension>
         <bloxlogic:tuple>
            <bloxlogic:member>
               [specify the member here]
            </bloxlogic:member>
            <bloxlogic:member>
             [specify another member here]
            </bloxlogic:member>
            ...
         </bloxlogic:tuple>
      </bloxlogic:tupleList>
   </bloxlogic:axis>
</bloxlogic:mdbQuery>
```

The <bloxlogic:tupleList> tag can also stand alone without being nested:

```
<bloxlogic:tupleList
   id=""
   tuplesRef="" />
```

The <bloxlogic:dimension> tag has one attribute:

```
<bloxlogic:dimension
   list="" />
```

The <bloxlogic:tuple> tag has one attribute:

```
<bloxlogic:tuple
   list="" />
```

The <bloxlogic:crossJoin> tag is a nested tag inside the <bloxlogic:axis> tag. It has no attribute. The <bloxlogic:member> tag also has no attributes.

MemberSecurityBlox

```
<bloxlogic:memberSecurity
   id=""
   cubeName=""
   dataBlox=""
   dataBloxRef=""
   dimensionName="" >
   <bloxlogic:memberSecurityFilter
      dimensionName=""
      memberName="" />
   <bloxlogic:memberSecurityFilter
      dimensionName=""
      memberName="" />
</bloxlogic:memberSecurity>
```

TimeSchemaBlox

```
<bloxlogic:timeSchema
   id=""
   dataBloxRef=""
   name=""
   today=""
/>
```

Blox Portlet Custom Tags

The Blox Portlet Tag Library has the following tags to help adding HTML markup on a portlet page based on ClientLink in the Blox UI Model:

- "The <bloxportlet:actionLinkDefinition> Tag" on page 474
- "The <bloxportlet:actionLink> Tag" on page 474
- "The <bloxportlet:PortletLinkDefinition> Tag" on page 474
- "The <bloxportlet:portletLink> Tag" on page 474
- "Nested <bloxportlet:parameter> Tag" on page 474

The <bloxportlet:actionLinkDefinition> Tag

```
<bloxportlet:actionLinkDefinition
   action="" />
```

The <bloxportlet:actionLink> Tag

```
<bloxportlet:actionLink
   action="" />
```

The <bloxportlet:PortletLinkDefinition> Tag

```
<bloxportlet:PortletLinkDefinition
   action="" />
```

The <bloxportlet:portletLink> Tag

```
<bloxportlet:portletLink
   action="" />
```

Nested <bloxportlet:parameter> Tag

This tag should be nested inside a `<bloxportlet:actionLinkDefinition>` tag or `<bloxportlet:PortletLinkDefinition>` tag:

```
   <bloxportlet:parameter
      name=""
      value="" />
```

Blox UI Custom Tags

To use the Blox UI modifier custom tags, import `bloxui.tld` as follows:

```
<%@ taglib uri="bloxuitld" prefix="bloxui"%>
```

The modifier custom tags include:

- Component customization tags
  - "The <bloxui:calculationEditor> Tag" on page 476
  - "The <bloxui:component> Tag" on page 476
  - "Custom Menu Tags" on page 478
  - "Custom Toolbar Layout Tag" on page 479

The <bloxui:actionFilter> Tag

```
<bloxui:actionFilter
   componentName=""
   filter="" />
```

The <bloxui:accessibility> Tag

```
<bloxui:accessibility
   screenReaderMode=""
   windowMenuEnabled="" />
```

The <bloxui:bottomN> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:bottomN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

The <bloxui:butterflyLayout> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:butterflyLayout
   addSeparatorColumns=""
   applyLayout=""
   description=""
   name=""
```

```
   position=""
   scope=""
   separatorWidth=""
   showOnLayoutMenu="" />
```

The <bloxui:calculationEditor> Tag

This tag has no attributes.

The <bloxui:clientLink> Tag

```
<!--Nested within a component customization tag-->

<bloxui:clientLink
   features=""
   link=""
   target=""/>
```

The <bloxui:component> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:component
   alignment=""
   bloxRef=""
   clickable=""
   disabled=""
   height=""
   name=""
   positionBefore=""
   style=""
   themeClass=""
   title=""
   tooltip=""
   valignment=""
   visible=""
   width="" >

      <bloxui:clientLink
         link=""
         target="" />

</bloxui:component>
```

The <bloxui:compressLayout> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:compressLayout
   applyLayout=""
   compressColumns=""
   compressRows=""
   description=""
   memberSeparator=""
   name=""
   showOnLayoutMenu="" />
```

The <bloxui:customAnalysis> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:customAnalysis
   analysis=""
   name="" />
```

The <bloxui:customLayout> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:customLayout
   applyLayout=""
   layout=""
   name=""
   showOnLayoutMenu=""/>
```

The <bloxui:eightyTwenty> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:eightyTwenty
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

The <bloxui:gridFilter> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridFilter
   filter="" />
```

The <bloxui:gridHighlight> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridHighlight
   applyLayout=""
   description=""
   includeData=""
   includeHeaders=""
   name=""
   scope=""
   selection=""
   showOnLayoutMenu=""
   style="" />
```

The <bloxui:gridSpacer> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:gridSpacer
   applyLayout=""
   axis=""
   description=""
   height=""
   locked=""
   name=""
   position=""
   scope=""
   showOnLayoutMenu=""
   style=""
   width="" />
```

The <bloxui:percentOfTotal> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:percentOfTotal
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
/>
```

The <bloxui:topN> Tag

```
<!--Nested within a PresentBlox or a GridBlox-->

<bloxui:topN
   description=""
   hideOthers=""
   membersToAnalyze=""
   name=""
   number=""
   preserveGrouping=""
   prompt=""
   showOtherSummary=""
   showRank=""
/>
```

The <bloxui:setProperty> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:setProperty
   name=""
   value="" />
```

The <bloxui:title> Tag

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui=title
   title=""
   style=""
   alignment=""/>
```

Custom Menu Tags

Tags for custom menu layouts include <bloxui:menu> and <bloxui:menuItem>:

The <bloxui:menu> and the <bloxui:menuItem> Tags

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:menu
   name=""
   bloxRef=""
   accesskey=""
   disabled=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible=""
   >
   <bloxui:menuItem
```

```
      name=""
      accesskey=""
      checkable=""
      checked=""
      disabled=""
      imageURL=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible=""
      >
      <bloxui:clientLink
         link=""
         target="" />
   </bloxui:menuItem>
</bloxui:menu>
```

Custom Toolbar Layout Tag

Tags for custom Toolbar layouts include `<bloxui:toolbar>` and `<bloxui:toolbarButton>`:

The <bloxui:toolbar> and Its Nested <bloxui:toolbarButton> Tags

```
<!--Nested within a PresentBlox, GridBlox, or ChartBlox -->

<bloxui:toolbar
   disabled=""
   bloxRef=""
   name=""
   positionBefore=""
   resourceName=""
   title=""
   tooltip=""
   visible="">
   <bloxui:toolbarButton
      checkable=""
      checked=""
      disable=""
      imageURL=""
      name=""
      positionBefore=""
      separator=""
      themeBasedImage=""
      title=""
      tooltip=""
      visible="" >
      <bloxui:clientLink
         link=""
         target="" />
   </bloxui:toolbarButton>
</bloxui:toolbar>
```

Relational Reporting Blox Custom Tags

Please see *Relational Reporting Developer's Guide* for custom tags for Blox supporting relational reporting.

Appendix B. Using the Alphablox XML Cube

The Alphablox XML Cube defines XML tags and attributes for representing query result sets returned from application data sources or DB2 Alphablox cubes. When a result set is transformed into an Alphablox XML Cube document, the document presents an open, predictable data structure with known elements, regardless of the layout of the underlying data source.

The Alphablox XML Cube presents a tree view of the data, as does the W3C XML DOM standard, where nodes correspond to data elements. The W3C XML DOM includes functions for manipulating document elements; Alphablox has extended the DOM to provide convenience methods particularly suited for manipulating analysis cube data. For more information on these extensions, see the com.alphablox.blox.xml package in the Javadoc documentation.

Application programmers can access the XML Cube document and process its data to implement custom logic or data layouts. This chapter explains the Alphablox XML Cube by using a familiar data representation often used in Alphablox applications.

Data Representation

Application programmers familiar with the DB2 Alphablox representation of an analysis cube result set will quickly understand the organization of the Alphablox XML Cube. This section reviews the key concepts of the DB2 Alphablox representation, using the following simple example:

Product	EAST	West	South	Market
Audio	12,460	15,507	0	27,967
Visual	33,138	26,605	24,565	84,308
Product	45,598	42,112	24,565	112,275

The result set includes descriptive elements (names of dimensions and members) and associated data values. A typical DB2 Alphablox representation, shown in the example, organizes the descriptive elements into row and column axes, and the data values into data cells. Note the following about the example:

- The Market dimension resides on the column axis, and includes three members: East, West, and South, as well as the Market rollup.
- The Product dimension resides on the row axis, and includes two members: Audio and Visual, as well as the Product rollup.
- Multiple dimensions can reside on the same axis. When this occurs, there is an implied grouping, with one dimension grouped within another.

- A *tuple* represents a set of members, one from each dimension on an axis. In the example, because there is only one dimension on each axis, each tuple represents a single member.
- Data values appear in cells at the intersection of tuples. For example, a value of 12,460 appears at the intersection of the Audio tuple and East tuple.

Note: In GridBlox or PresentBlox, unused dimensions reside on the Other axis. In the Alphablox XML Cube, each unused dimension resides on a separate slicer axis.

The next section explains how to render a query result set into XML format.

Sample Alphablox XML Document

Below is the example result set rendered as an XML document. In some cases, line breaks have been added for readability.

```
<?xml version="1.0"?>

<!DOCTYPE cube SYSTEM '/alphablox/AnalysisServer/xml/dtd/cube.dtd'>

<cube>
   <bloxInfo>
     <bloxID>15</bloxID>
     <bloxName>MyDataBlox</bloxName>
     <appName>MyXMLDoc</appName>
   </bloxInfo>
   <data>
   <slicer>
     <slicerDimension name="Period">Period</slicerDimension>
     <slicerMember name="Period" gen="1"
                   leaf="false">Period</slicerMember>
     </slicer>
     <slicer>
       <slicerDimension name="Accounts">Accounts</slicerDimension>
       <slicerMember name="Accounts" gen="1"
                     leaf="false">Accounts</slicerMember>
     </slicer>
     <slicer>
       <slicerDimension name="Scenario">Scenario</slicerDimension>
       <slicerMember name="Scenario" gen="1"
                     leaf="false">Scenario</slicerMember>
     </slicer>
  <axis name="columns" index="0">
       <dimensions>
        <dimension name="Market" index="0">Market</dimension>
       </dimensions>
       <tuple index="0">
        <member name="East" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">East</member>
       </tuple>
       <tuple index="1">
        <member name="West" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">West</member>
</tuple>
<tuple index="2">
        <member name="South" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">South</member>
</tuple>
<tuple index="3">
        <member name="Market" index="0" gen="1" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Market</member>
</tuple>
  </axis>
```

```
  <axis name="rows" index="1">
<dimensions>
        <dimension name="Product" index="0">Product</dimension>
</dimensions>
<tuple index="0">
        <member name="Audio" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Audio</member>
</tuple>
<tuple index="1">
        <member name="Visual" index="0" gen="2" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Visual</member>
</tuple>
<tuple index="2">
        <member name="Product" index="0" gen="1" spanInHierarchy="1"
            spanIndexInHierarchy="0" leaf="false">Product</member>
</tuple>
  </axis>
  <cells>
    <row>
      <column>
        <cell>13438.0</cell>
      </column>
      <column>
        <cell>22488.0</cell>
      </column>
      <column>
        <cell>0.0</cell>
      </column>
      <column>
        <cell>35926.0</cell>
      </column>
    </row>
    <row>
      <column>
        <cell>33138.0</cell>
      </column>
      <column>
        <cell>40351.0</cell>
      </column>
      <column>
        cell>24565.0</cell>
      </column>
      <column>
        <cell>98054.0</cell>
      </column>
    </row>
    <row>
      <column>
        <cell>46576.0</cell>
      </column>
      <column>
        <cell>62839.0</cell>
      </column>
      <column>
        <cell>24565.0</cell>
      </column>
      <column>
        <cell>133980.0</cell>
      </column>
    </row>
  </cells>
 </data>
</cube>
```

Alphablox XML Tags

DB2 Alphablox uses the XML tags described in this section to represent the elements in a query result set returned from an application data source. The tags support analysis cubes with an unlimited number of axes.

As with all XML tags, the following rules apply to the Alphablox tags:

- An XML document must begin with the XML declaration:
  `<?xml version="1.0"?>`
- There can be one and only one root element in an XML document. All other tags that define the document's content are contained within the root element. The following tags define the root element for the Alphablox XML Cube:
  `<cube>...</cube>`
- Tags can nest but cannot overlap. For example, the following structure is valid:
  `<bloxInfo><bloxName>myBlox</bloxName></bloxInfo>`
  but the following structure is not valid:
  `<bloxInfo><bloxName>myBlox</bloxInfo></bloxName>`

The tags are listed in their order of appearance in an XML document.

XML Tag	
`<?xml version="1.0"?>`	Identifies an XML document and names the W3C specification that it uses. The Alphablox XML Cube uses the 1.0 specification.
`<!DOCTYPE cube...>`	Identifies the document type (by naming its root element) and specifies the associated DTD file.
`<cube> </cube>`	Identifies the document root element for the Alphablox XML Cube. There can be one and only one cube element in an XML document. All the other elements are contained within it.
`<bloxInfo> </bloxInfo>`	Provides information about this Blox.
`<bloxID> </bloxID>`	Identifies this Blox instantiation (the value is automatically provided by DB2 Alphablox).
`<bloxName> </bloxName>`	Identifies the Blox used for this instantiation (the value is taken from the `bloxName` property).
`<appName> </appName>`	Identifies the application (the value is taken from the Blox `applicationName` property).
`<data> </data>`	Identifies the entire data area of the XML document, including axis definitions and data cells.
`<slicer> </slicer>`	Identifies the area of the XML document that defines a slicer axis. A slicer axis provides a "slice through" the cube. With OLAP data sources, each dimension that does not appear in the data area is placed on a separate slicer axis, along with one of its members.
`<slicerDimension> </slicerDimension>`	Names the dimension on the slicer axis.
`<slicerMember> </slicerMember>`	Names the member in a slicer dimension.

<axis> </axis>	Identifies the axis type (typically column or row) and defines its contents. The five named axes (in low-to-high sequence) are column, row, page, chapter, and section. An axis can also be referred to as Axis[index], where index is in the range of 0 to N. For example, one way to refer to the row axis is Axis[1]; the way to refer to the first unnamed axis, is Axis[5].
<dimensions> </dimensions>	Identifies an area within the <axis> </axis> tags where the dimensions on the axis are named.
<dimension> </dimension)	Identifies a specific dimension.
<tuple> </tuple>	Identifies an element containing a set of all the dimension members on one axis.
<member> </member>	Identifies a member that belongs to a tuple.
<cells> </cells>	Identifies the area of the XML document that contains data values (as opposed to dimension and member headings).
<axisCells> </axisCells>	Identifies an unnamed axis (also referred to as Axis[5] through Axis[N]) and includes its data cells.
<section> </section>	Identifies a section axis (also referred to as Axis[4]) and includes its data cells.
<chapter> </chapter>	Identifies a chapter axis (also referred to as Axis[3]) and includes its data cells.
<page> </page>	Identifies a page axis (also referred to as Axis[2]) and includes its data cells.
<row> </row>	Identifies a row axis (also referred to as Axis[1]) and includes its data cells.
<column> </column>	Identifies a column axis (also referred to as Axis[0]) and includes its data cells.
<cell> </cell>	Identifies a data value at the intersection of tuples.

Alphablox XML Tag Attributes

Alphablox XML tags use the following attributes. Remember that indexes are 0-based.

Attribute	Description
gen	Identifies the generation level of this element within its data hierarchy (specifically, the level of a member within its dimension). For example, in the Product dimension, Product has a gen value of “1”, Audio and Visual have a gen value of “2”, and VCR and TV have a gen value of “3”.
index	Identifies the position of this element in a series of like elements. For example, the following lines indicates that this is the first tuple: <tuple index=″0″>...</tuple>

leaf	Specifies if this is a leaf node (true) or not (false).
name	Provides a unique name for this element. For example, the following line provides a name (as well as a data value) for a dimension element: <dimension name=″memberName″ index=″0″>AliasName</dimension>
span	Identifies the number of tuples that a member spans. For example, a member named Qtr1 would have a span of “3” (for January, February, and March). For MemberElement, use spanInHierarchy rather than span.
spanInHierarchy	Identifies the number of tuples that a member spans within a hierarchy defined by the same root parent. For example, a member named March sould have a spanInHierarchy value of 3. For MemberElement, use spanInHierarchy rather than span.
spanIndex	Indicates the zero-based position of this member in a series of spanned members. For example, January would have a spanIndex of “0”; February, “1”; and March, “2”. For MemberElement, use spanIndexInHierarchy rather than spanIndex.
spanIndexInHierarchy	Indicates the zero-based position of this member in a series of spanned members, but relative to the root parent in its hierarchy. For example, if April has a spanIndex of 3, but occurs within the column Qtr2, the spanIndexInHierarchy value would be 0. For MemberElement, use spanIndexInHierarchy rather than spanIndex.

XML Data Islands

An XML data island is a block of valid XML code embedded inside an HTML document. Data islands enable programmers to script against the XML document without having to load it (through script or the <OBJECT> tag). Currently, XML data islands are supported only in Microsoft Internet Explorer 5.5 and later.

Definition Syntax

The syntax for defining an inline data island in a page appears below. Note the use of the <XML> and </XML> tags:

```
<XML ID="DataIslandID">
  <XMLDATA>
    <DATA>TEXT</DATA>
  </XMLDATA>
</XML>
```

For example, the following lines define a data island with three data values:

```
<XML ID="MyDataIsland">
  <dataSources>
    <dataSource name="DB2">IBM DB2 OLAP Server 8.1</dataSource>
    <dataSource name="MSOLAP">Microsoft OLAP Services 7.0</dataSource>
    <dataSource name="Essbase">Hyperion Essbase 6.5</dataSource>
  </dataSources>
</XML>
```

The contents of a data island can also reside in an external file. Use the following syntax to include an external XML file as a data island:

```
<XML SRC="http://<server>/MyXmlFile.xml"></XML>
```

XMLDocument Property

The XMLDocument property returns the root node of the inline or external XML data island. Programmers can use the standard XML DOM to navigate the data island from this root. For example, the following function returns all the data from MyDataIsland.

```
function returnXMLData(){
   return document.all("MyDataIsland").XMLDocument.nodeValue;
}
```

The following syntax is also valid. Using the example from "Definition Syntax" on page 486, the line returns a value of "IBM DB2 OLAP Server 8.1":

```
MyDataIsland.XMLDocument.documentElement.childNodes.item(0).text
```

DataBlox as an XML Data Island

The following lines define a standard DataBlox as a data island:

```
<XML ID="MyDataBlox">
   <blox:data id="MyDataBlox"
      dataSourceName = "qcc">
      query = "!"
      render = "XML">
   </blox:data>
</XML>
```

By setting the `render` attribute to `XML` causes the DataBlox result set to be rendered into XML format. When the page is processed, the lines defining the Blox are replaced by the rendered XML lines.

Elsewhere in the page, a JavaScript function could gain access to the contents of the data island through syntax like the following:

```
MyDataBlox.getCube.getUniqueName
```

Appendix C. DB2 Alphablox XML Cube DTD

Similar to a database schema, a *Document Type Definition* (DTD) defines the data structures that can occur in a document and the sequence in which they can occur.

Knowing a document's structure enables programmers to write code that can traverse the document, extract specific values from it, and perform operations on the values.

DTD Syntax Notes

The following markup appears in the DB2 Alphablox XML Cube DTD:

Element Type Declaration

Names an element and specifies its children.

Syntax

```
<!ELEMENT name (childElement1, childElement2,...childElementN)>
```

Usage

A regular expression character (+, *, or ?) appended to the name of a child element specifies how many of that child element the parent can contain. The absence of one of these characters indicates that the parent element can contain one and only one of that child element.

+	The parent element can contain one or more of the named child element
*****	The parent element can contain zero or more of the named child element
?	The parent element can contain zero or one of the named child element

For example, a line like the following:

```
<!ELEMENT data (slicer*, axis+, cells)>
```

would specify that the data element can have zero or more slicer elements, one or more axis elements, and only one cells element.

Attribute List Declaration

Names an element and specifies its attributes. For each attribute, specifies its name, data type, and required or optional presence.

Syntax

```
<!ATTLIST element-name
        childElementName1 dataType #REQUIRED
        childElementName2 dataType #REQUIRED
        childElementNameN dataType #REQUIRED
>
```

Usage

For example, the following lines:

```
<!ATTLIST dimension
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>
```

specify that the dimension member has two required attributes (name and index), both of which contain plain character data.

Data Types

Associated with an element name, specifies the type of data permitted in the element.

Usage

The DB2 Alphablox XML Cube DTD uses the following two data types:

#PCDATA (Parsed Character Data): plain (non-markup) text that may contain entity references. For example, the string & should be parsed to yield an ampersand sign.

CDATA (Character Data): plain (non-markup) text that does not include entity references. For example, a less-than sign (<), quotation marks (“), or an ampersand (&) are treated as plain text and not parsed.

Tip: A complete discussion of DTD syntax is beyond the scope of this guide.

DTD Elements

For an explanation of each DTD Element that appears in the listing, see “Alphablox XML Tags” on page 484.

DTD Listing

The remainder of this section is a listing of the DB2 Alphablox XML Cube DTD.

```
<!ELEMENT cube (bloxInfo, data)>

<!ELEMENT bloxInfo (bloxID, bloxName, appName)>

<!ELEMENT bloxID (#PCDATA)>
<!ELEMENT bloxName (#PCDATA)>
<!ELEMENT appName (#PCDATA)>

<!ELEMENT data (slicer*, axis*, cells)>

<!ELEMENT slicer  (slicerDimension, slicerMember)>

<!ELEMENT slicerDimension (#PCDATA)>

<!ATTLIST slicerDimension
        name CDATA #REQUIRED
>

<!ELEMENT slicerMember (#PCDATA)>

<!ATTLIST slicerMember
      name CDATA #REQUIRED
      gen CDATA #REQUIRED
      leaf CDATA #REQUIRED
```

```
>

<!ELEMENT axis (dimensions,tuple*)>

<!ATTLIST axis
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>

<!ELEMENT dimensions (dimension*)>

<!ELEMENT tuple (member*)>

<!ATTLIST tuple
        index CDATA #REQUIRED
>

<!ELEMENT dimension (#PCDATA)>

<!ATTLIST dimension
       name CDATA #REQUIRED
       index CDATA #REQUIRED
>

<!ELEMENT member (#PCDATA)>

<!ATTLIST member
       name CDATA #REQUIRED
       gen CDATA #REQUIRED
       span CDATA #REQUIRED
       spanIndex CDATA #REQUIRED
       spanInHierarchy CDATA #REQUIRED
       spanIndexInHierarchy CDATA #REQUIRED
       index CDATA #REQUIRED
       leaf CDATA #REQUIRED
>

<!-- for zero axis, we have cell value only -->
<!ELEMENT cells (axisCells* | section* | chapter* | page*| row*| column*|
                cell)>
<!ELEMENT axisCells (axisCells+ | section+)><!ATTLIST axisCells
       name CDATA #REQUIRED
>

<!ELEMENT section (chapter+)>
<!ELEMENT chapter (page+)>
<!ELEMENT page (row+)>
<!ELEMENT row (column+)>
<!ELEMENT column (cell)>
<!ELEMENT cell (#PCDATA)>
```

Appendix D. Examples Cross References

This chapter contains the additional code examples and cross references to examples in this book:

Category	Example
BookmarksBlox	• "Example 1: Getting a count of all bookmarks" on page 56 • "Example 2: Getting the properties set for a Bookmark" on page 56 • "Example 3: Getting a list of bookmarks that match the specified criteria" on page 58 • "Example 4: Creating a bookmark using BookmarksBlox API" on page 59 • "Example 5: Using server-side bookmarkLoad event filter" on page 60 • "Example 6: Getting a bookmark's query when it is loaded" on page 61
Blox Form Tag Library and FormBlox	• "A CheckBoxFormBlox Example" on page 312 • "A CubeSelectFormBlox Example" on page 314 • "A DataSourceSelectFormBlox Example" on page 316 • "DimensionSelectFormBlox Examples" on page 319 • "An EditFormBlox Example" on page 320 • "A MemberSelectFormBlox Example" on page 324 • "A RadioButtonFormBlox Example" on page 325 • "A SelectFormBlox Example" on page 328 • "A TimePeriodSelectFormBlox Example" on page 332 • "A TreeFormBlox Example" on page 338
Business Logic Blox and the Blox Logic Tag Library	• "An CrossJoin Example" on page 347 • "An MDBQueryBlox Example" on page 349 • "A MemberSecurityBlox Example" on page 351 • "A TimeSchemaBlox Example" on page 353 • "A Sample TimeSchema for IBM DB2 OLAP Server or Hyperion Essbase Data Sources" on page 354 • "A Sample TimeSchema for Microsoft Analysis Services Data Sources" on page 354

Category	Example
Blox UI Tag Library	• "Component Tag Examples" on page 370 • "bottomN Tag Examples" on page 375 • "Menu Tag Examples" on page 398 • "Toolbar Tags Examples" on page 404 • "Toolbar Tags Examples" on page 404
ChartBlox	• "Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method" on page 496 • Dynamically setting chart type using a SelectFormBlox: "A SelectFormBlox Example" on page 328
CommentsBlox	• "Example 1: Enabling cell commenting" on page 144 • "Example 2: Specifying Field to Sort On and Sort Order" on page 144 • "Example 3: Accessing Cell Comments Using MDBResultSet" on page 145 • "Example 4: Adding a CommentAddedEvent Listener" on page 146
Data Calculation	• "Example 1: Walk Through a Relational Result Set" on page 494 • "Example 1: Adding a calculated member named Profit at the end of the Measures dimension" on page 175 • "Example 2: Specifying the position of the calculated member" on page 175 • "Example 3: Adding a generation number and scope" on page 175 • "Example 4: Replacing missing or null values with the value 0" on page 175 • "Example 5: Calculations involving members from different dimensions" on page 176 • "Example 6: Adding ranking" on page 177 • "Example 7: Adding a separate ranking within each group" on page 177 • "Example 8: Adding running totals within each group" on page 178
MemberFilterBlox	• "Example 1: Filtering Members for All Available Dimensions" on page 262 • "Example 2: Filtering Members for Specified Dimensions Only" on page 263 • "Example 3: Filtering Members for One Dimension Only" on page 263

Example 1: Walk Through a Relational Result Set

```
<%--  RDBResultSet.jsp
----  Example page to illustrate RDB ResultSet Methods
----
----  Walk a server-side RDB ResultSet and output the column
---   metadata information and the first and last rows of data.
 --%>

<%-- Import the Alphablox taglib --%>
<%@ taglib uri="bloxtld" prefix="blox" %>
<%-- Import the packages for accessing the server-side RDBResultSet--%>
<%@ page import="com.alphablox.blox.data.rdb.*" %>

<%-- creates sqlTypes variable & imports java.sql.Types & java.util.Hashtable--
%>
<%@ include file="SQLTypes.jsp"%>

<blox:data id="relationalDB"
  dataSourceName = "qcc-mssql"
  query = "SELECT * FROM qcc WHERE Sales > 9000 ORDER BY Week_Ending,
Product_Family_Code"
>
</blox:data>
```

```
<%
  RDBResultSet rs = (RDBResultSet) relationalDB.getResultSet();

  // Get the schema details
  ResultColumn[] cols       = rs.getColumns(); // column Metadata
  int[]          types      = rs.getTypes();
                              // jdbc/sql data types in the rs
  int            colCount   = cols.length;
                              // num cols in result set

  // each row of data is returned as an array of objects; use types
  // to determine their data types
  Object[]       firstRow   = null;
  Object[]       lastRow      = null;
  int            rowsRead   = 0;

  // iterate through the rows incrementing the row counter and
  // saving the first and last rows of data
  while( rs.hasMoreRows() )
  {
    rowsRead++;
    if( rowsRead == 1 )
    {
      firstRow  = rs.getNextRow( false );
    }
    lastRow = rs.getNextRow( false );
  }
%>

<html>
<head>
    <title>Relational JSP</title>
    <blox:header/><%-- Blox header tag for standard js and style inclusions --%>
</head>

<body>
<br>
The column count is: <b><%= colCount %></b><br />
The row count is: <b><%= rowsRead %></b><br />
The columns are: <br />

<table border="1" cellspacing="0" cellpadding="0">
  <tr>
    <th>Col Name</th>
    <th>Type</th>
    <th>Type Name</th>
    <th>First Row</th>
    <th>Last Row</th>
  </tr>
  <%
    // Display the column names, their types, typeNames, and
    // the first and last row of data
  for( int i = 0; i < colCount; i++ )
  {
    out.write("\t<tr>");
    out.write("\t\t<td>" + cols[i].getName() + "</td>" ); // Col Name
    out.write("\t\t<td>" + String.valueOf(types[i]) + "</td>"); // Type
    // the names of the sql types is in the sqlTypes hashtable created in
SQLTypes.jsp
    out.write("\t\t<td>" + String.valueOf( sqlTypes.get( new Integer( types[i]
) ) ) + "</td>" ); // Type Name
    out.write("\t\t<td>" + String.valueOf(firstRow[i]) + "</td>"); //First Row
    out.write("\t\t<td>" + String.valueOf(lastRow[i]) + "</td>"); //Last Row
    out.write("\t</tr>");
  }
```

```
%>
</table>
</body>
</html>
```

The following code is the SQLTypes.jsp file referenced in the JSP file above:

```
<!-- SQLTypes.jsp
---- Helper page to create a hashtable with all of the SQL data types
---- 2002.03.28  -  YRL & REK
----
 -->

<!-- Imports for standard Java classes used -->
<%@ page import="java.sql.Types.*" %>
<%@ page import="java.util.Hashtable" %>

<%
    Hashtable sqlTypes = new Hashtable();

    sqlTypes.put( new Integer( java.sql.Types.ARRAY ), "ARRAY" );
    sqlTypes.put( new Integer( java.sql.Types.BIGINT ), "BIGINT" );
    sqlTypes.put( new Integer( java.sql.Types.BINARY ),"BINARY" );
    sqlTypes.put( new Integer( java.sql.Types.BIT ), "BIT" );
    sqlTypes.put( new Integer( java.sql.Types.BLOB ), "BLOB" );
    sqlTypes.put( new Integer( java.sql.Types.CHAR ), "CHAR" );
    sqlTypes.put( new Integer( java.sql.Types.CLOB ), "CLOB" );
    sqlTypes.put( new Integer( java.sql.Types.DATE ), "DATE" );
    sqlTypes.put( new Integer( java.sql.Types.DECIMAL ), "DECIMAL" );
    sqlTypes.put( new Integer( java.sql.Types.DISTINCT ), "DISTINCT" );
    sqlTypes.put( new Integer( java.sql.Types.DOUBLE ), "DOUBLE" );
    sqlTypes.put( new Integer( java.sql.Types.FLOAT ), "FLOAT" );
    sqlTypes.put( new Integer( java.sql.Types.INTEGER ), "INTEGER" );
    sqlTypes.put( new Integer( java.sql.Types.JAVA_OBJECT ) ,
                            "JAVA_OBJECT"   );
    sqlTypes.put( new Integer( java.sql.Types.LONGVARBINARY ),
                            "LONGVARBINARY" );
    sqlTypes.put( new Integer( java.sql.Types.LONGVARCHAR ),
                            "LONGVARCHAR"   );
    sqlTypes.put( new Integer( java.sql.Types.NULL ), "NULL" );
    sqlTypes.put( new Integer( java.sql.Types.NUMERIC ), "NUMERIC" );
    sqlTypes.put( new Integer( java.sql.Types.OTHER ), "OTHER" );
    sqlTypes.put( new Integer( java.sql.Types.REAL ), "REAL" );
    sqlTypes.put( new Integer( java.sql.Types.REF ), "REF" );
    sqlTypes.put( new Integer( java.sql.Types.SMALLINT ), "SMALLINT" );
    sqlTypes.put( new Integer( java.sql.Types.STRUCT ), "STRUCT" );
    sqlTypes.put( new Integer( java.sql.Types.TIME ), "TIME" );
    sqlTypes.put( new Integer( java.sql.Types.TIMESTAMP ),
                            "TIMESTAMP");
    sqlTypes.put( new Integer( java.sql.Types.TINYINT ), "TINYINT" );
    sqlTypes.put( new Integer( java.sql.Types.VARBINARY ),"VARBINARY");
    sqlTypes.put( new Integer( java.sql.Types.VARCHAR), "VARCHAR");
%>
```

Example 2: Set Chart Properties on the Server Using the bloxAPI.call() Method

```
<%--  chartSelect.jsp
----  Example page to illustrate how to use the call method to
----  execute some server-side code.
--%>

<!-- Import the taglib -->
<%@ taglib uri = "bloxtld" prefix = "blox"%>
<html>
<head>
```

```
   <title>Change Repository Values</title>
   <blox:header />
</head>

<!-- The JavaScript function that passes the chart type selected and
---- calls another JSP page (setSelection.jsp) on the server to set
---- the chart type.
--%>

<script language="JavaScript">
   function setChartChoice(ChrtType) {
      bloxAPI.call("setSelection.jsp?chart="+ChrtType);
   }
</script>

<body>
<blox:present id = "myPresent"
   height = "400"
   width = "600"
   >

   <blox:chart
      chartType = "Vertical Bar, Side-by-Side">
   </blox:chart>
   <blox:data
      dataSourceName = "QCC-Essbase"
      useAliases = "true"
      query = "<ROW ('All Products')  <ICHILD 'All Products' <SYM
         <COLUMN ('All Time Periods') <Ichild '2001' !"
   >
   </blox:data>
</blox:present>
<br>
Select a chart type:
<form name = form1>
   <input type="radio" name="chartSelect" value="Bar"
      onclick="setChartChoice(value);"> Bar
   <input type="radio" name="chartSelect" value="Line"
      onclick="setChartChoice(value);"> Line
   <input type="radio" name="chartSelect" value="Pie"
      onclick="setChartChoice(value);"> Pie
</form>
</body>
</html>
```

The JSP file called has the following code:

```
<%--  setSelection.jsp
----  Called by chartSelect.jsp to set the chart type to the one
----  selected.
--%>

<!-- Import the taglib -->
<%@ taglib uri="bloxtld" prefix="blox"%>

<%-- Reference the instance of PresentBlox created in chartSelect.jsp
--%>

<blox:present id="myPresent" />
<%
String chartChoice = request.getParameter("chart");
if (chartChoice != null && chartChoice.trim().length() != 0) {
   myPresent.getChartBlox().setChartType(chartChoice);
   };
%>
```

Example 3: Use the server-side ChartPageListener to set the desired data format on the chart when the chart filter is changed

This example demonstrates how to use the server-side event listener to set the format for Y1Axis (`setY1FormatMask()`) in order to maintain the correct formatting that has been set in the GridBlox when users change the filter in the chart. In this case:

- The Scenario dimension is on the page axis. Both Actual and Variance % have cell formats specified.
- Using the `addEventListener()` method, specify the instance of a ChartPageListener object (`CPListener()`) to call when users change the filter on the chart.
- `CPListener` implements the `ChartPageListener` interface.
- Get the member selected. In this particular example, since the member Variance % has a "%" in the name, we test to see the return string ends with "%". If it does, we set the `Y1FormatMask` accordingly.

```
<%@ page import="com.alphablox.blox.ChartBlox,
                 com.alphablox.blox.event.*,
                 com.alphablox.blox.uimodel.core.MessageBox,
                 com.alphablox.blox.uimodel.BloxModel,
                 com.alphablox.blox.ServerBloxException"%>
<%@ taglib uri="bloxtld" prefix="blox"  %>
<html>
<head>
   <blox:header />
</head>
<body>
<blox:present id="present" height="400" width="600" >
   <blox:grid defaultCellFormat="#,##0.00;[red](#,##0.00)" >
      <blox:cellFormat index="1" format="#,##0.00;[red](#,##0.00)"
         scope="{Scenario:Actual}" ></blox:cellFormat>
      <blox:cellFormat index="2" format="#,##0.00%;[red](#,##0.00%)"
         scope="{Scenario:Variance %}" ></blox:cellFormat>
   </blox:grid>
   <blox:chart chartType="bar" autoAxesPlacement="false"
      filter="Scenario" XAxis="All Time Periods" legend="All Locations" >
   </blox:chart>
   <blox:data dataSourceName="QCC-Essbase"
      query="{OUTALTNAMES} <ROW (\"All Locations\") <ICHILD \"All Locations\"
<COLUMN (\"All Time Periods\", \"Scenario\") <SYM \"Jan 01\" \"Feb 01\"
\"Mar 01\" \"Apr 01\" \"May 01\" \"Jun 01\" \"Actual\" \"Variance %\" !">
   </blox:data>
   <%-- adds a ChartPageFilter to the ChartBlox --%>
   <% present.getChartBlox().addEventListener(new
CPListener(present.getBloxModel()));%>

</blox:present>
</body>
</html>

<%!
  public class CPListener implements ChartPageListener
  {
    BloxModel model;
    public CPListener (BloxModel model) {
       this.model = model;
    }
    public void changePage(ChartPageEvent cpe) {
      ChartBlox blox = cpe.getChartBlox();
      try {
        if (cpe.getSelection().endsWith("%")) {
          String msg = new String("Setting format mask to be a percentage");
```

```
          blox.setY1FormatMask("#%");
           MessageBox msgBox = new MessageBox(msg, "Chart Page Filter",
MessageBox.MESSAGE_OK, null);
          model.getDispatcher().showDialog(msgBox);
        }
        else {
          String msg = new String("Setting format mask to be currency");
          blox.setY1FormatMask("$#K");
          MessageBox msgBox = new MessageBox(msg, "Chart Page Filter",
MessageBox.MESSAGE_OK, null);
          model.getDispatcher().showDialog(msgBox);        }
      } catch (ServerBloxException e) {
        e.printStackTrace();
      }
    }
  }
%>
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation, Licensing, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation, J46A/G4, 555 Bailey Avenue, San Jose, CA 95141-1003 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	DB2	DB2 OLAP Server
DB2 Universal Database™	WebSphere®	

Alphablox and Blox are trademarks or registered trademarks of Alphablox Corporation in the United States, other countries, or both.

Microsoft, Windows®, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Index

D

E

F

G

H

I

R

S

T

U

V

W

X

Y

Program Number: 5724-L14

Printed in USA

SC18-9435-02

Spine information:

IBM DB2 Alphablox Developer's Reference Version 8.4